Exploring the Explainable Aspects and Performance of a Learnable Evolutionary Multiobjective Optimization Method

GIOVANNI MISITANO, University of Jyväskylä, Finland

Multiobjective optimization problems have multiple conflicting objective functions to be optimized simultaneously. The solutions to these problems are known as Pareto optimal solutions, which are mathematically incomparable. Thus, a decision maker must be employed to provide preferences to find the most preferred solution. However, decision makers often lack support in providing preferences and insights in exploring the solutions available.

We explore the combination of learnable evolutionary models with interactive indicator-based evolutionary multiobjective optimization to create a learnable evolutionary multiobjective optimization method. Furthermore, we leverage interpretable machine learning to provide decision makers with potential insights about the problem being solved in the form of rule-based explanations. In fact, we show that a learnable evolutionary multiobjective optimization method can offer advantages in the search for solutions to a multiobjective optimization problem. We also provide an open source software framework for other researchers to implement and explore our ideas in their own works.

Our work is a step toward establishing a new paradigm in the field on multiobjective optimization: *explainable and learnable multiobjective optimization*. We take the first steps toward this new research direction and provide other researchers and practitioners with necessary tools and ideas to further contribute to this field.

CCS Concepts: • **Theory of computation** → **Bio-inspired optimization**; • **Computing methodologies** → **Rule learning**; • **Information systems** → **Decision support systems**; • **Software and its engineering** → **Open source model;**

Additional Key Words and Phrases: Multiobjective optimization, evolutionary multiobjective optimization, learnable evolutionary models, explainable artificial intelligence

ACM Reference format:
Giovanni Misitano. 2024. Exploring the Explainable Aspects and Performance of a Learnable Evolutionary Multiobjective Optimization Method. *ACM Trans. Evol. Learn.* 4, 1, Article 4 (February 2024), 39 pages.
https://doi.org/10.1145/3626104

1 INTRODUCTION

The field of **Multiobjective Optimization (MOO)** [Miettinen 1999; Sawaragi et al. 1985; Steuer 1989] specializes in solving problems with multiple conflicting objective functions with various tradeoffs. The objective functions depend on decision variables, which are often real valued. MOO problems have many mathematically incomparable optimal solutions known as **Pareto Optimal**

Author's address: G. Misitano, Faculty of Information Technology, University of Jyväskylä, P. O. Box 35 (Agora), Jyväskylä, Finland, 40014; e-mail: giovanni.a.misitano@jyu.fi.

2688-3007/2024/02-ART4
https://doi.org/10.1145/3626104

(PO) solutions. Because the solutions are incomparable without additional information, we must employ the aid of a **Decision Maker (DM)** with domain expertise to identify the most preferred solution. The DM provides preferences, which are utilized to find the best possible solution among the PO ones. MOO problems are an omnipresent kind of problem in the real world, and it is therefore important that we are not only able to understand and solve these problems, but to also provide the adequate decision support to DMs so that they may find the most preferred solution to make the best possible decisions.

The challenge of solving MOO problems is twofold. First, there is the computational and mathematical aspect of finding PO solutions. There are various approaches in this regard, such as scalarization-based (see, e.g., [Miettinen 1999]) and **Evolutionary Algorithms (EAs)** (see, e.g., [Branke et al. 2008; Coello et al. 2007]). By scalarizing a MOO problem, its objective functions are aggregated into one and the problem is reduced to a single-objective optimization problem, a scalarized problem, using some scalarization function. Then, the scalarized problem can be optimized utilizing appropriate, single-objective optimizers. The scalarization function must be selected carefully to guarantee the resulting solution to be PO [Miettinen 1999; Sawaragi et al. 1985]. In turn, EAs work by iteratively evolving a population of solution candidates, gradually improving the population. EAs are based on heuristics inspired by Darwinian evolution and cannot guarantee the Pareto optimality of the solutions, but they are able to produce multiple, near-optimal solutions, whereas by scalarizing a MOO problem one can find a single, but more accurate solution—under certain assumptions even guaranteed to be PO [Miettinen 1999; Sawaragi et al. 1985]. EAs may also be used to solve scalarized MOO problems. In addition to scalarization-based and population-based methods, other types of set approximation methods like the ones discussed in the work of Talbi et al. [2012] exist as well, but are not considered in our current work.

The second challenge of solving MOO problems lies in how to incorporate and utilize preferences provided by the DM to find the most preferred solution. Preferences can be incorporated in the solution process in three ways [Hwang and Masud 1979; Miettinen 1999]: before (*a priori*) or after (*a posteriori*) the optimization process, or interactively during the optimization process. Methods implementing the latter are known as interactive MOO methods [Miettinen 1999]. The main drawbacks of providing preferences before or after the optimization process are, respectively, the possible lack of knowledge the DM has about the characteristics of the solutions available, such as the ranges of the objective function values and the tradeoffs between them, and the (often) overwhelming amount of available solutions to choose from. Naturally, if a DM is very familiar with the MOO problem being solved, then these drawbacks are diminished. However, interactive MOO methods allow the DM to explore the problem by providing preferences iteratively and seeing what kind of solutions are available. This allows the DM to *learn* about the MOO problem [Miettinen et al. 2008]. Interactive MOO methods require little in terms of prior knowledge the DM has about the problem, and they also vary in the type of preferences they accept, how the preferences are incorporated, and what kind of information is shown to the DM [Afsar et al. 2021; Miettinen et al. 2016; Xin et al. 2018]. That being said, DMs often lack support when providing preferences in interactive MOO, which is still an open research question in the study of interactive methods [Belton et al. 2008; Wang et al. 2016].

Recently, it has been argued that interactive MOO methods can be seen as black-boxes by the DM [Misitano et al. 2022]: preferences go in the method which then generates new solutions, but how? To remedy this, *explainability* was explored as a potential solution to provide the DM with additional insights about how a scalarization-based interactive method maps preferences to generate new solutions in the work of Misitano et al. [2022]. It was shown that explainability can have the potential to support the DM in providing new preferences and reach their most preferred PO solution in fewer iterations.

In our present work, we will explore explainability in the context of EAs—specifically *indicator-based* EAs—for solving MOO problems combining them with **Learnable Evolutionary Models (LEMs)** [Michalski 2000]. LEMs are a class of optimization algorithms that combine EAs with **Machine Learning (ML)** models. The idea is to use heuristics based on Darwinian evolution in a so-called *Darwinian mode* to improve a population of solutions, then occasionally switch to a so-called *learning mode* where ML is utilized to learn a hypothesis describing what characterizes a desired solution. This hypothesis can then be used to further generate potentially desired solutions to add to the population. A LEM algorithm then switches between a Darwinian and learning mode iteratively. In our current work, we utilize an explainable and interpretable ML model in a LEM to solve MOO problems and utilize the explanations to provide new insights to the DM about the solutions computed for a MOO problem by explaining the connection between decision variables and objective function values for solutions that are congruent with preferences expressed by the DM. As a proof of concept, we develop a new interactive MOO method based on the idea of LEMs, which utilizes an indicator-based EA and an explainable ML model.

It is safe to assume for a DM to be mainly interested in the objective function values of a MOO problem [Miettinen 1999]—it is often the case that the number of objective functions in a problem is much lower than the number of decision variables, and the problem is assumed to be modeled in a manner where the objective functions represent important criteria to the DM. However, a DM can often also be interested in some of the decision variable values of the computed solutions [Hakanen et al. 2011, 2013; Kania et al. 2022]. We show that explainability can be leveraged in this aspect as well by providing DMs with additional insights about the ranges of the decision variable values in the proximity of a preferred solution. From a design perspective, where the decision variable values of a problem can also matter, insight about the variables can lead to better decision making, such as by helping DMs discover new designs and practical considerations of the solutions [Deb and Srinivasan 2006; Ray et al. 2022; Smedberg and Bandaru 2022]. Our proposed approach should be of special interest to more technical DMs, such as engineers, who are more accustomed with the implications of different decision variable values in addition to being familiar with the objective function values.

We are the first to explore the explainable aspects in solving MOO problems with LEMs. We provide simple, rule-based explanations to help a DM make a connection between the variables and objective functions in solutions found near a point of interest (from the perspective of the DM) to a MOO problem. To achieve this goal, we implement our own explainable method for solving MOO problems based on LEMs and a simple indicator-based EA. We also provide open source software for others to implement and explore their own similar methods. In our work, we explore the contribution of the ML model used to the performance of our method in finding approximate PO solutions. Based on these contributions, we pave the way toward a new subfield in MOO: **Explainable and Learnable Evolutionary Multiobjective Optimization (XLEMOO)**.

To summarize, the contributions of our present work are the following:

(1) Following the core idea of LEMs, we combine an indicator-based **Evolutionary Multiobjective Optimization (EMO)** with an interpretable ML model to produce an XLEMOO method.
(2) We explore the performance of our **Learnable Evolutionary Multiobjective Optimization (LEMOO)** method in its ability to find optimal solutions. We study this by varying parameters that are specific to the LEM part of our approach.
(3) We demonstrate and discuss the potential benefits of the explanations produced by the interpretable ML model to a DM in our LEMOO method in a showcase as a proof of concept.
(4) We provide an open source Python framework—the XLEMOO framework—to implement and explore further ideas related to XLEMOO.

The purpose of our work is to show that by combining interpretable ML models with EAs into a new method, we can potentially achieve an overall better performance of the method (compared to the EA part alone) and be able to provide valuable explanations regarding the connection between decision variables and objective functions values to a DM. To the best of our knowledge, in the context of applying LEMs to MOO, we are also the first to publish the software used and the results of the experiments conducted in a manner that allows others to reproduce our work and build on top of it promoting the openness and renewal of science.

This article is structured as follows. In Section 2, we present the necessary theory and background information, and give references to the most relevant works related to our work. In Section 3, we present the general structure of a XLEMOO method and briefly discuss the XLEMOO framework. In Section 4, we explore the impact of ML in our LEMOO method from a performance point of view by varying parameters specific to the LEM part of our method. We then move to Section 5, where we present a showcase on how the insights emerging from the explainability of our LEMOO method can help a DM in an interactive MOO context, where a DM provides preferences iteratively during the optimization process. We present the results and implications of the experiments and the showcase in Section 6, where we also discuss the further potential of explainability in the context of LEMOO. The article is then concluded with Section 7, where we draw the main conclusions of our work and propose future research directions in the field of XLEMOO.

2 BACKGROUND

In this section, we provide the background information necessary for the rest of the article. We start with Sections 2.1 and 2.2, where we briefly present the basics of (interactive) MOO and EMO focusing on indicator-based methods, respectively. In Section 2.3, we present the main ideas behind LEMs and how these have been utilized in MOO in the past. Furthermore, we briefly discuss the basics of **Explainable Artificial Intelligence (XAI)** and ML in Section 2.4. We conclude with Section 2.5, where we will review the most notable past works in which explainability has been explored in the context of MOO.

2.1 Concepts of MOO

A MOO problem can be defined as follows:

$$\begin{aligned} \text{minimize} \quad & F(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})), \\ \text{s.t.} \quad & \mathbf{x} \in X \subset \mathbb{R}^n, \end{aligned} \tag{1}$$

where $f_1, f_2, \ldots, f_k$ are ($k \geq 2$) objective functions, $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$ is a *decision vector* of n decision variables, and X is the set of feasible decision vectors. We assume the objective functions to be real valued—that is, $f_i : \mathbb{R}^n \to \mathbb{R}^k$. The feasible set is often defined by box-constraints on the decision variables (i.e., upper and lower bounds) or constraint functions, or both. A feasible solution to a MOO problem (1) is one that belongs to the feasible set X. When all k objective functions are evaluated at some feasible decision vector $\mathbf{x}$, the result is an *objective vector* $\mathbf{z}$, with components $z_i = f_i(\mathbf{x})$ for $i = i, \ldots, k$, which forms the set of feasible objective vectors Z. We assume all objective functions to be minimized without loss in generality—a function to be maximized can be converted to a minimized function by multiplying it by -1.

While objective vectors cannot be fully compared on a mathematical basis alone, we can define the concept of *dominance* to help us order objective vectors. Suppose we have two feasible decision vectors $\mathbf{x}^1$ and $\mathbf{x}^2$ with respective objective vectors $\mathbf{z}^1$ and $\mathbf{z}^2$. Now, $\mathbf{x}^1$ is said to dominate $\mathbf{x}^2$ if, and only if, $z_i^1 \leq z_i^2$ for all $i = 1, \ldots, k$ and $z_j^1 < z_j^2$ for at least *some* $j = 1, \ldots, k$. We can then define the set of PO solutions as the subset of feasible solutions, which when evaluated will result in a

set of mutually nondominated solutions so that no other feasible solution exists that dominates any of the solutions in the set. The set of objective vectors resulting from evaluating the set of PO solutions is the set of PO objective vectors Z^{Pareto}, which we refer to as a *Pareto set*.

To characterize the Pareto set, we define the concepts of *ideal* and *nadir* points. When minimizing each objective function, the ideal point $\mathbf{z}^*$ represent the lower bounds of the Pareto set. The ideal point can be computed by minimizing each objective function in a MOO problem (1) individually. Similarly, the nadir point $\mathbf{z}^{\text{nadir}}$ represents the upper bounds of the Pareto set. However, computing the nadir point would require knowledge on the whole and true extent of the Pareto set, which is generally unknown. Several methods exist to approximate the nadir point (e.g., [Benayoun et al. 1971; Deb and Miettinen 2009; Deb et al. 2010]).

As mentioned, a MOO problem (1) can be scalarized using a *scalarization function* $s : \mathbb{R}^k \rightarrow \mathbb{R}$ that transforms the original MOO problem into a single-objective optimization problem. The achievement scalarizing function [Wierzbicki 1980, 1982] is an example of a scalarization function. The single-objective optimization problem resulting from utilizing the achievement scalarizing function is defined as

$$\min_{\mathbf{x}\in X} s^{\text{ASF}}(F(\mathbf{x}), \bar{\mathbf{z}}, \mathbf{z}^{**}, \mathbf{z}^{\text{nadir}}) = \min_{\mathbf{x}\in X} \max_{i=1,\ldots,k} \left[\frac{f_i(\mathbf{x}) - \bar{z}_i}{z_i^{\text{nadir}} - z_i^{**}} \right] + \rho \sum_{i=1}^{k} \frac{f_i(\mathbf{x})}{z_i^{\text{nadir}} - z_i^{**}}, \tag{2}$$

where $\bar{\mathbf{z}} = (\bar{z}_1, \ldots, \bar{z}_k)$ is a *reference point*, a vector consisting of *aspiration levels*, which are provided by a DM; $\mathbf{z}^{**}$ is a *utopian* point, defined as $z_i^{**} = z_i^* - \varepsilon$, for $i = 1 \ldots k$, where ε is a small positive scalar; and ρ is also a small positive scalar value. The aspiration levels in the reference point represent objective function values a DM wishes to achieve.

The scalarization function in (2) is also able to incorporate the preferences of a DM in the optimization process in the form of aspiration levels in the reference point. As said, the aspiration levels represent objective function values the DM wishes to achieve. When solved, (2) results in a PO solution with an objective vector that is projected to the closest PO solution according to the Tchebycheff [Miettinen 1999] distance of its components to the aspiration levels in the reference point. By utilizing different reference points $\bar{\mathbf{z}}$ in (2), we can find different solutions. The summation term in (2) is known as an *augmentation term* that assures the Pareto optimality of the solution found—actually, so-called proper Pareto optimality (for details, see, e.g., [Miettinen 1999; Wierzbicki 1982]). Many scalarizing functions exist with different properties (see, e.g., [Miettinen and Mäkelä 1999, 2002]).

2.2 Evolutionary Multiobjective Optimization

EMO methods [Branke et al. 2008; Coello et al. 2007] take inspiration from Darwinian evolution to evolve a set (a *population*) of solutions (referred to as *individuals*) concurrently over multiple generations, gradually improving the overall *fitness* of the population. EMO methods can be divided into three subcategories: indicator-based [Zitzler and Künzli 2004], domination-based [Deb et al. 2002], and decomposition-based [Zhang and Li 2007] methods. Although EMO methods can generate multiple solutions at the same time, there is no guarantee for the solutions to be PO due to the heuristic nature. This is why EMO methods can be considered to generate, at best, only approximations of PO solutions and sets. Different EMO methods are more suitable for different problems and may accommodate for different preference information as well. For recent reviews on different EMO methods, see, for example, the work of Antonio and Coello [2017] and Chugh et al. [2019].

We focus our current study on an indicator-based method, which evolves a population of solutions over generations utilizing three simple evolutionary operators: crossover, mutation, and

selection. The crossover, or mating, operator combines different individuals to form new ones, the mutation operator applies random mutations to the individuals, and the selection operator is used to select the individuals with a good fitness. In indicator-based methods, a quality (or performance) indicator is used in the selection step to select individuals from the population with good fitness values. The selected individuals then continue to the next generation.

By properly selecting the quality indicator, we can determine what kind of solutions the EMO algorithm will strive to find. Usually, indicators measure the quality of the whole set of found solutions by either comparing the set to another set or by producing an absolute measure of the quality of an individual set [Emmerich and Deutz 2018]. The fitness of a single solution is then determined by its contribution to the overall value of the indicator. When preference information is available from a DM, the fitness of single solutions can be determined directly by utilizing a scalarizing function, such as the one shown in (2).

When incorporating preference information in the indicator used, it is possible to focus the search for solutions in an indicator-based EMO method to a subset of the approximate PO solutions that is of special interest to a DM. In this case, there is no need to try to approximate the whole PO set. Furthermore, when the DM is allowed to change the preferences and provide them iteratively between the executions of the EMO method, we have an interactive method. From a decision-making perspective, interactive methods are important since they allow the DM to learn about the available solutions and the feasibility of their own preferences [Miettinen et al. 2008]. The details of the proposed interactive indicator-based EMO method that incorporates explainability are further fleshed out in Sections 3 and 4.

2.3 Learnable Evolutionary Models

LEMs are a special breed of EAs where the heuristic nature of EAs is augmented with the more deterministic nature of ML methods. Proposed originally in the work of Michalski [2000], LEMs are claimed to help an evolutionary process converge faster, as in fewer iterations, to an optimal solution. Leveraging the power of ML is made possible in EAs because of the large number of individuals generated during an evolutionary process. This allows an ML model to distinguish between *good* and *bad* individuals and formulate a hypothesis on how to further generate new, usually good individuals. This process can boost the overall search for optimal solutions by diversifying the population and by finding better individuals. In the original work [Michalski 2000], it was also suggested that the evolutionary part of a LEM may be replaced by ML, but we will not explore this aspect in our current work. More details on how a LEM works are given in Section 3.1.

In the past, LEMs have been applied for solving MOO problems. We next describe some of these works. Perhaps the earliest work in which LEMs have been applied for solving MOO problems was presented in by Jourdan et al. [2005], where a LEM was applied to solve a MOO problem in water system design. The authors were able to show that a LEMOO method offered notable performance advantages when compared to NSGA-II [Deb et al. 2002]. In the work of Moradi and Mirzaei [2016], a LEMOO method was merged with a knowledge base to successfully automate complementary metal-oxide semiconductor (CMOS) analog circuit design. The inclusion of a LEM was shown to considerably decrease the number of circuit evaluations. Likewise, in the work of Moradi [2018], LEMs were applied for solving a multiobjective robot path-planning problem, and it was shown that the approach led to a higher hypervolume and overall set coverage of the evolved population when compared to other similar methods. The same author further applied their approach to a vehicle routing problem with stochastic demand with similar results [Moradi 2019]. Solving MOO vehicle routing problems was also explored in the work of Niu et al. [2021] with results indicating better performance in terms of computation time and the quality of solutions when compared to the state-of-the art EMO methods.

Although the listed works have been successful in applying LEMs to solve MOO problems, none of the works provide the software for implementing and applying LEMs to solve MOO problems, nor do they provide the data of the experiments that were run. This not only raises issues with the replicability of the results but also makes it very cumbersome for other researchers and practitioners to apply LEMOO in their own works.

In our work, we provide an open source software framework for implementing and experimenting with LEMOOs as an extension of the DESDEO software framework [Misitano et al. 2021] for MOO, and we make the data and means to reproduce the experiments shown in later sections of our paper openly available. We also provide insights for researchers and practitioners alike to start building and experimenting with their own implementations of LEMOO methods.

2.4 ML and XAI

ML [Bishop 2006] is the study and application of software that can learn from data or its surroundings, or both, to make conclusions about new observations. Often, ML is used to predict new information, examples of which include the prediction of skin cancer in patients [Esteva et al. 2017] or the fault diagnosis in rotating machinery [Liu et al. 2018], for instance. However, when used in decision support, ML can be problematic because of its black-box nature. The opacity resulting from this black-box nature is not desirable if we wish to make transparent and justifiable decisions. It is an unfortunate fact that the ML model complexity correlates positively with its predictive power [Gunning and Aha 2019], but exceptions to this rule do exist [Lipton 2018]. The more complex a model is, the harder it is generally to interpret and understand.

The field of XAI [Kamath and Liu 2021; Molnar 2022] focuses on the study of the explainable aspects of **Artificial Intelligence (AI)**, including ML, and the development of XAI methods. Focusing on explainable ML, there are two main ways that explainability can be incorporated into an ML model. First, there is the *model agnostic* approach, which focuses on methods that are able to explain *any* ML model. This is achieved by just observing the input and output of an ML model and trying to build an understanding on the mapping between these two. Examples of model agnostic explainability are SHAP values [Lundberg and Lee 2017] and LIME [Ribeiro et al. 2016]. Another approach to incorporate explainability in ML is using inherently interpretable ML models. These include, but are not limited to, rule- and tree-based models. Since rules and trees are easy to interpret, there is no need to utilize external tools; instead, the rules and trees can be directly observed to build an understanding on how the ML model makes predictions. Nevertheless, sometimes even inherently interpretable ML models can become too complex to be feasibly interpreted by a human. This is the case with decision trees with a depth in the hundreds, for instance. As argued in the work of Lipton [2018], model interpretability is not a monolithic concept; it also depends on the specific model itself. For reviews on recent advancements in explainable ML and XAI methods, see, for example, the work of Arrieta et al. [2020] and Linardatos et al. [2020].

In our work, we utilize skope-rules [Goix et al. 2020] as an ML model to learn highly interpretable, and thus explainable, rule sets. Skope-rules utilize an ensemble of decision trees to learn with a high precision instances of different classes. The trees are then reduced into simple and interpretable `IF...THEN...` rules. For example, if skope-rules have been applied to learn and predict a binary classification (classes 0 or 1) on two real-valued variables a_1 and a_2 ranging from 1 to 10, then a learned rule set could look, for instance, as follows:

RULE 1: IF $a_1 < 8.2$ AND $a_2 > 1.3$ THEN PREDICT 1,
RULE 2: IF $a_1 > 5.5$ AND $a_1 < 6.3$ AND $a_2 < 1.4$ THEN PREDICT 1,
RULE 3: IF $a_2 > 1.1$ THEN PREDICT 1.

Notice that the class of 0 is assumed to hold when no rules apply to an observed sample. Each rule is also associated with a *precision*, which measures how many samples in the training dataset each rule applies to. This precision ranges from 0 to 1, where 1 means a rule applies to all samples and 0 that it applies to none. For example in the preceding first rule, when a_1 is less than or equal to 8.2, and a_2 is greater than 1.3, then the predicted class for the instance is 1. Skope-rules have been applied successfully in the literature to extract accurate rules to increase the explainability of ML applications, for example, in the work of Bologna [2021] and Narteni et al. [2021].

From a societal perspective, explainability is an important concept that must be taken into account when utilizing ML in decision support. The European Union's General Data Protection Regulation (GDPR) includes the notion for humans to have a *right to an explanation* (Recital 71[1]) in the context of algorithmic decision support, including the use of AI and ML [Goodman and Flaxman 2017].

2.5 Explainability in MOO

Since MOO methods serve primarily as decision support tools, it is no surprise that explainability has recently begun to attract attention in MOO as well. Generally, we may speak of *explainable MOO* when talking about MOO methods with incorporated explainability. In what follows, we briefly review how explainability has been applied to MOO in the literature.

In the work of Wang et al. [2016], a MOO approach was utilized to build a movie recommendation system that accounts for both accuracy of the prediction and individual diversity oriented at users. By taking into account the recommended items' content (the movie genre) when defining the lists' diversity, the recommended lists were claimed to be explainable to the user. Explainability was not, however, explored in the MOO method itself but rather in the problem formulation.

A framework to explain different policies in a multiobjective probabilistic planning setting was presented in the work of Sukkerd et al. [2018]. The explanations of the policies were meant to increase end users' confidence in the different policies. The underlying preference structure leading to each policy was also highlighted. The policy justification system was twofold: first, it described which quality attributes had been considered in each policy, and second, it could argue why a particular policy had been generated. The framework was able to argue why a certain policy was superior, or at least not worse than another one. It achieved this goal by providing the user qualitative rather than numerical information.

So-called *relationship explainable* MOO was explored in the work of Zhan and Cao [2019] through an actor-critic reinforcement learning method. The method was able to learn in a quantifiable way the inter-relationships between different objective functions by a novel concept, *marginal weights*. By being quantifiable, the authors claimed that the relationships could also be explainable. This work focused on the study of the tradeoffs between the conflicting objectives in a MOO problem.

Belief-rules were utilized in the work of Misitano [2020] to model the preferences of a DM during an interactive MOO process. The preferences were modeled as a utility function, which was learned through reinforcement learning by utilizing a belief-rule-based system as an ML model. Although the rules learned by the model could be explained, the explainable aspect of the preference model was only discussed as a potential of the proposed method.

Modeling the preferences of a DM was also explored by Corrente et al. [2021], where a dominance-based rough set approach was used to model preferences based on the results of pairwise comparisons conducted by the DM. The preferences of the DM were then highly explainable

[1] https://www.privacy-regulation.eu/en/r71.htm

because they were modeled based on human interpretable rules. In this work, an interactive EMO method was studied.

Lastly, in the work of Misitano et al. [2022], SHAP values [Lundberg and Lee 2017] were utilized to explain the relationship between preferences and computed solutions in an interactive MOO method. Based on the explanations produced, suggestions were also formulated to aid the DM in providing further preferences to help them achieve a desired goal of improving a certain objective function value in the solution. A small case study was also conducted to validate the explanations and suggestions, and it was found that the suggestions helped the DM reach his best solution in fewer iterations and with less guessing.

In our work, we provide to a DM insights about the connection between decision and objective vectors. However, we do not claim to be the first ones to explore this connection. This concept has been studied previously and has been shown to provide important information to a DM when exploring and selecting solutions in a MOO context. For example, the concept of *innovization* coined by Deb and Srinivasan [2006] leverages EMO to not only find optimal solutions but to also help unveil new design principles in engineering problems. This unveiling is based on building an understanding of the relationship between decision variables and objective functions. More recent examples of connecting the decision variables and the objective functions to support better decision making, can be found, for example, in the work of Ray et al. [2022] and Nagar et al. [2022].

Related to innovization, data-mining approaches have also been explored in the past to find design patterns and to discover knowledge in MOO problems. Although these approaches have not been explicitly referred to as *explainable*, these approaches address similar issues that explanations do. These data-mining approaches have been surveyed in the work of Bandaru et al. [2017a], and more recent advancements have been discussed in another work by Bandaru et al. [2017b]. Data mining for knowledge discovery has also been studied very recently in an interactive MOO setting as well in the work of Smedberg and Bandaru [2022].

To the best of our knowledge, our work is the first to build a connection between decision and objective vectors by utilizing explainability in particular emerging from leveraging interpretable ML in a LEM to interactively solve MOO problems. As the explanations vary depending on the ML model used, the software framework we provide makes it readily possible to explore other ML models as well leading to different explanations, which may serve different DMs better than others.

3 LEMOO AND THE XLEMOO FRAMEWORK

In this section, we present the general structure of a LEMOO method in Section 3.1. We then discuss the potential of explainability in LEMOO methods in Section 3.2. We conclude this section by discussing the proposed XLEMOO software framework to implement LEMOO methods with explainable ML models in Section 3.3. In the following sections, Sections 4 and 5, we utilize our own LEMOO method implementation built according to the framework discussed here.

3.1 LEMOO

In LEMOO, a population of solutions to a MOO problem (1) is evolved in two modes: in a *Darwinian mode* and in a *learning mode*. In both modes, the population can be iteratively evolved multiple times before switching to the other mode. The rules to determine when the mode should be switched can be defined in multiple ways. The simplest is to iterate each mode for a fixed number of times. Another option is to switch modes when a certain condition is met—for example, the fitness of the overall population has improved past a certain threshold. In our present work, for simplicity, we will switch modes after a fixed number of times. Let N^D (Darwinian) and N^L (learning) be the numbers of iterations each mode is iterated before switching to the other

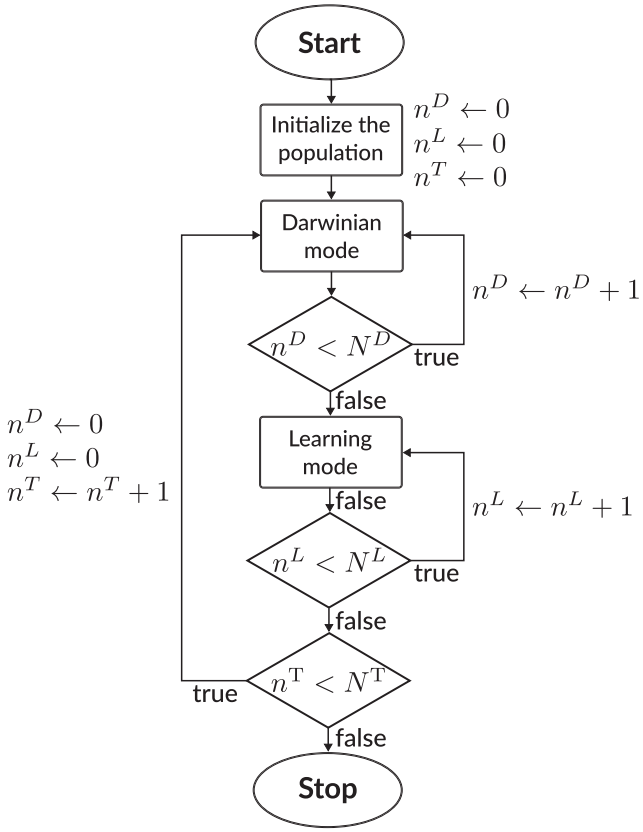

Fig. 1. The basic flowchart of a LEMOO method.

mode. Furthermore, let N^T be the total number of iterations the LEMOO method is iterated over. Iterating a LEMOO method once means evolving the population in a Darwinian mode N^D times and in a learning mode N^L times. This concept is further clarified in Figure 1. According to Figure 1, a LEMOO method starts by initializing the population of solutions. This can be done by utilizing Latin hypercube sampling [McKay et al. 2000], for instance. Whether a LEMOO method starts in a Darwinian mode or a learning mode can also be changed, but we assume that each iteration of a LEMOO method always starts in a Darwinian mode. Notice that if $N^D = 0$ or $N^L = 0$, the respective mode is skipped completely. How often a LEMOO method switches from a Darwinian mode to a learning mode is defined by the learning mode *switching frequency*. A switching frequency defines the number of iterations in a Darwinian mode to be completed before switching to a learning mode. For example, a switching frequency of 10 means that after 10 iterations in a Darwinian mode, the LEMOO method switches to a learning mode, which is equivalent to setting $N^D = 10$. Next, we further discuss the Darwinian and learning modes.

3.1.1 Darwinian Mode. In a Darwinian mode, the population is evolved utilizing an EA method. In our present work, we apply an indicator-based EMO that first performs a crossover operation on the population, then mutates it, and finally selects according to an indicator the best individuals to continue to the next generation. We utilize the scalarizing function in (2) as an indicator and utilize it directly to compute the fitness of individuals in a population. We have chosen the scalarizing function in (2) as the indicator because it incorporates preference information in the form of a reference point, which can be provided by a DM. Thus, we are able to focus the evolution-

ary process to find solutions that are interesting from the perspective of the DM. The scalarizing function in (2) also allows us to fully order solutions in a population based on the fitness values. Moreover, when utilizing (2) to compute the fitness, dominated solutions will always get a worse fitness when compared to the fitness of nondominated solutions. We are also able to avoid deteriorating effects by utilizing the scalarizing function in (2) as an indicator in the selection step of our indicator-based method, which effectively implements elitism. It is also important to emphasize that in our approach, we do not try to approximate the whole set of PO solutions. Instead, we focus the search on a small subset of PO solutions that are close to a reference point provided by a DM.

Other indicators may also be chosen to compute the fitness in the described approach, but we have utilized the scalarizing function in (2) throughout our current work. The EA used in Darwinian mode may not necessarily have to be an indicator-based EMO. Nonetheless, whichever EA is selected, the learning mode in LEMOO is characterized by its heuristic nature. The main flow of a Darwinian mode is depicted in Figure 2 (left).

3.1.2 Learning Mode. The idea of a learning mode is to employ an ML method to learn characteristics of good solutions. Following the original work [Michalski 2000], ML is utilized for binary classification. Individuals are classified into high-performing and low-performing groups: an *H-group* and an *L-group*, respectively (which we call an *H/L splitting*). These groups are then used as training data for the ML model. This phase is known as *hypothesis forming*. After the ML method has been trained to learn a hypothesis, it is then used to *instantiate* new individuals. The best of the new individuals are then combined with the H-group of individuals to form a new population. Individuals may also be discarded when forming a new population if the size of the population is desired to be kept constant. Depending on the ML model employed, generating new individuals can be either straightforward, such as in the case of rule-based and tree-based ML methods, or it can be more complicated, such as in the case of neural networks. Other than binary classification of the solutions is also possible. For instance, an ML model could be used to learn rules to sort solutions into the different fronts in an EMO algorithm like NSGA-II [Deb et al. 2002]. The model can then be used to generate new solutions with different ranks, possibly improving the variety of the solutions, for example.

The crucial difference on how ML is utilized in LEMOO is that ML is *not* used to predict information; instead, it is used to *describe* and generate new data. One important choice with binary classification to be made when implementing a LEMOO method is to decide how the H- and L-groups are chosen. One option, and the one utilized in our current work, is to pick a percentile of the best and worst individuals based on their fitness values and form the H- and L-groups, respectively. We have utilized the fitness values computed based on the scalarizing function in (2), which reflects the preferences of a DM. Furthermore, we assume the top and bottom percentage to be always equal and represented by an *H/L splitting ratio* or just *H/L split*. The main flow of a learning mode is depicted in Figure 2 (right).

As seen in Figure 2, both Darwinian and learning modes act on a population to improve it. In a Darwinian mode, new solutions are produced based on heuristics and the best solutions are selected based on the solutions' fitness values. However, in learning mode, a hypothesis is learned that describes good individuals and it is used to generate new, potentially good individuals. When the rule learned is interpretable by humans, such as in rule-based ML, we have also access to explanations and explainability.

3.2 The Potential for Explainability in LEMOO

When properly selected, the ML model in a LEMOO method not only gives a boost to the performance of the method but also makes the method explainable for a DM. Explainability is what can really elevate LEMOO methods as decision support tools when compared to traditional EMO

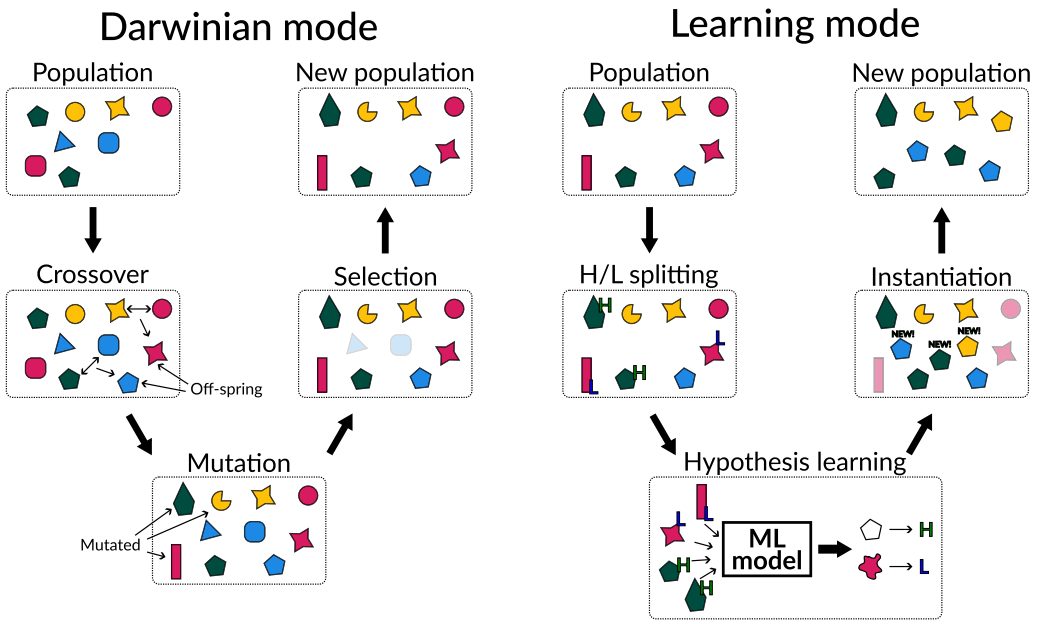

Fig. 2. A pictorial representation of how the two modes of LEM, Darwinian mode and learning mode, act on the population of solutions to create a new population.

methods. For example, by selecting an inherently interpretable ML model, such as a rule-based model, the rules generated by the model can be used to provide the DM with additional insights about the solutions in a population. In our current work, we have decided to utilize skope-rules as an interpretable form of ML because of its low computational demands and the interpretability of the generated rules. Usually, the final population has the solutions with the best fitness values, which makes the ML model trained in the last iteration of a LEMOO method of particular interest from an explainability perspective. For example, if a rule-based ML model has been trained to classify individuals in a population into H- and L-groups based on the individuals' decision vectors, the learned rules can then be shown to a DM to provide additional information about what kind of decision variable values constitute a good solution. In our present work, we have utilized the scalarizing function in (2), which incorporates a DM's preferences. This means that the information extracted from the rules will describe solutions that pertain to the DM's preferences in the decision space of the MOO problem being solved. As mentioned in Section 1, the DM may be expected to be mainly interested in the objective function values of the solutions, but from a design and engineering perspective, the decision variable values can play an important role in the final decision making. We will explore the explanations produced by our LEMOO method in an interactive setting in Section 5 to further clarify the added benefits of explainability. We will also see that the information extracted from the rules can support a DM in providing new preferences addressing an open question in the field of interactive MOO: How to support DMs in providing preferences in the context of interactive MOO methods? How these rules may be generated is discussed in Section 4.

We believe that, when leveraged, the explainable aspects of LEMOO methods can provide similar insights to a DM between the decision variables and objective functions as discussed in Section 1 (c.f. innovization). However, to the best of our knowledge, in none of the past works has the potential for explainability been explored in LEMOO methods to provide a DM with

additional insights about the decision vectors in the computed solutions. Furthermore, past works exploring LEMOO have not provided software or other means for researchers and practitioners alike to start exploring and implementing LEMOO methods. Through the experiments conducted in Section 4, we will provide novel insights about LEMOO methods for others to consider when building their own LEMOO methods, and we will show how explainability can be combined with LEMOO methods to provide DMs with important insights about the MOO problem being solved. Thus, we give rise to the new breed of XLEMOO methods, where explainability and LEMOO have been combined to provide DMs with additional information to support better decision making when applying EMO methods. To help others get started with their own implementations and applications of XLEMOO, we provide the open source software framework, *the XLEMOO framework*, as a bedrock for researchers and practitioners to lean on in their future endeavors. This framework is discussed next.

3.3 The XLEMOO Framework

The XLEMOO (software) framework allows users to readily build and apply LEMOO methods to solve MOO problems as described in Section 3.1. The fitness function, the evolutionary operators in a Darwinian mode, and ML model in a learning mode can be easily switched or replaced with custom implementations. The user has also access to the various parameters present in Darwinian and learning modes. Although the framework is primarily built to accustom indicator-based methods in a learning mode, switching to another kind of EMO method is possible with moderately little effort. We have also taken care to document the framework, which eases its use. The framework is implemented in the Python language, which is a widely used language in data analysis, and the framework is aimed at users with a moderate proficiency in the language.

We have implemented the XLEMOO framework as open source software extending the DESDEO framework [Misitano et al. 2021]. The XLEMOO framework is available on GitHub[2] and Zenodo [Misitano 2023a]. To our knowledge, this is the first time a framework for implementing LEMOO methods has been made available to the public. The framework is also suitable for solving single-objective optimization problems. On top of the documentation, we have also provided a Jupyter [Kluyver et al. 2016] notebook for users to get quickly started utilizing the framework.[3]

In the following section, we give a detailed example of our own LEMOO method implemented in the described XLEMOO framework. We have utilized the implementation to run experiments and explore the effects of a learning mode on the performance of a LEMOO method, and also showcase the potential of the explanations emerging from the learning mode of the LEMOO method when the ML model utilized is interpretable.

4 IMPLEMENTATION AND EXPERIMENTS

In this section, we present in more detail the implementation of our LEMOO method utilizing the XLEMOO framework discussed in Section 3. The implementation and experimental setup are presented in Section 4.1. Then, in Section 4.2, we use our implementation to study the effects of the switching frequency and the H/L split in a learning mode on the performance of our method in numerical experiments. Last, we explore the potential for explainability offered by our method and discuss how the explanations (i.e., rule sets) have been generated in Section 4.3. The implementation discussed in this section is later utilized in Section 5 as an interactive MOO method, where the added benefit of explainability to a DM is showcased. The reader interested in the practical benefits offered by our method can skip this section and go to Section 5.

[2] https://github.com/gialmisi/XLEMOO/tree/article_v1.1
[3] https://xlemoo.readthedocs.io/en/article_v1.1/notebooks/Showcase.html

4.1 General Setup

We implemented our LEMOO method utilizing the framework discussed in Section 3.3. The following parameters were chosen in the method's Darwinian and learning modes. This is the same implementation that is utilized later in Section 5.

4.1.1 Darwinian Mode. We implemented a simple indicator-based EMO algorithm with the achievement scalarizing function (2) as the indicator (i.e., the fitness function), which computes a fitness value for computed solutions. The lower the fitness value, the closer the solution is to a supplied reference point $\bar{\mathbf{z}}$, reflecting how well a solution aligns with the preferences of a DM represented by the reference point. In (2), ρ and ε were set to 10^{-6} throughout the experiments. The population size was set to be $N^{\text{pop}} = 50$ to keep computation times feasible. The initial population was initialized using Latin hypercube sampling to achieve a uniform initial coverage of different decision vectors. Three evolutionary operators, as shown in Figure 2 (left), were chosen. For the crossover operator, simulated binary crossover [Deb et al. 1995] was utilized with a crossover probability of 1.0 and a crossover distribution parameter equal to 30; bounded polynomial mutation [Deb et al. 1995] was used as the mutation operator with a mutation probability of $\frac{1}{n}$, where n is the number of decision variables in a MOO problem, and a mutation distribution parameter equal to 20; the selection operator was defined such that after crossover and mutation, the N^{pop} best individuals were chosen to proceed to the next generation, keeping the population size constant in each generation. The population from each generation was saved into an archive. With these operators and their parameters, our method seemed to achieve acceptable results consistently, which motivated their choice. We utilized the EMO operators implemented in the `desdeo-emo` package [Misitano et al. 2021], version 1.4.1.

4.1.2 Learning Mode. We chose skope-rules implemented in the `imodels` Python package [Singh et al. 2021] as the ML model utilized in a learning mode to learn a rule set for binary classification. Our choice was motivated by the computationally fast training of skope-rules, and the interpretability of the generated rules (see Section 4.3 for examples of these rules). The minimum rule precision considered was set to be 0.1, the maximum number of estimators to be learned was set to 30, and we chose to bootstrap both samples and features. The rest of the skope-rules' parameters were kept at their default values as defined in the `imodels` package, version 1.3.6. We chose these values because we found them to work well from a performance perspective in our case. To train the ML model, we considered the populations from all past generations computed before the current learning mode iteration to maximize the available training data. Based on the fitness values of the individuals, we chose a top (best fitness values) and bottom (worst fitness values) percentage of the individuals from all past populations. Individuals with the best fitness values were close to the reference point in (2), whereas individuals with the worst values were farther away. When forming the H- and L-groups, only unique solution, based on their decision variable values, were considered. This was done to boost the learning performance of the ML model. Then, the ML model was trained to classify individuals into either a high-performing or a low-performing group.

The trained skope-rules model was then used to instantiate $\gamma^{\text{inst}} \times N^{\text{pop}}$ high-performing new individuals, where γ^{inst} is an *instantiation factor*, which was set to 10. We chose a relatively high instantiation factor to increase the changes of finding new high-performing solutions during the instantiation. The individuals were instantiated based on the rule sets learned by the ML model. Rules in the rule set were weighted based on their precision when instantiating new individuals: rules with a higher precision were used more often to generate new individuals than rules with a lower precision. Based on a rule, the decision variable values of instantiated individuals were

Table 1. Computed Payoff-table for the Vehicle Crash Worthiness Problem

Vehicle crash worthiness problem			
Minimized	f_1	f_2	f_3
f_1	**1,661.71**	8.30	0.071
f_2	1,675.45	**6.14**	0.26
f_3	1,692.02	10.56	**0.042**

The first column indicates which objective has been minimized on each row. The minimum values in each column for each problem are in bold, whereas the maximum values are underlined.

randomly generated according to the upper and lower limits extracted from the rule. If a rule did not describe an upper or lower limit for a variable, the upper and lower bound based on box constraints of the variable were used instead, respectively. We did not consider problems with function constraints on the variables, but implementing an instantiation strategy to account for function constraints as well is possible in the XLEMOO framework. The new instantiated individuals and the high-performing group were then combined, and the selection operator from the learning mode was used to select the best individuals, keeping the population size constant. The new population generated in the learning mode was also saved in the population archive.

4.1.3 MOO Problem Setup. In the experiments conducted, we considered two real-life based MOO problems. The first one is the *vehicle crash worthiness problem* discussed in Section 5. We will present the results for the vehicle crash worthiness problem later in this section but have included the results for the second problem in the appendix. We made this choice to maintain the clarity of the main text and to not inflate the number of figures present in the text. We utilized the problems as implemented in the `desdeo-problem` package [Misitano et al. 2021], version 1.4.6.

The second problem considered in the experiments, the *car-side impact problem* [Jain and Deb 2013], extended with a fourth objective from the work of Tanabe and Ishibuchi [2020], optimizes the crash safety of a car in case of a side impact. The original problem had three objectives to be minimized: the mass of the vehicle, the pubic force experienced by a passenger in case of a crash, and the average velocity experienced by the V-pillar withstanding the impact load from a crash. The fourth objective is the sum of the 10 constraints the decision variables of the problem are subject to. The seven decision variables of the problem, which model the thickness of the various car parts (in millimeters) that contribute to the crash safety of the vehicle, are subject to the following box constraints: $x_1 \in [0.5, 1.5], x_2 \in [0.45, 1.35], x_3 \in [0.5, 1.5], x_4 \in [0.5, 1.5], x_5 \in [0.875, 2.625], x_6 \in [0.4, 1.2]$, and $x_7 \in [0.4, 1.2]$.

Because the achievement scalarizing function (2) requires information about a problem's ideal and nadir points, we computed the approximations of these points by utilizing our LEMOO method. We ran the method solely in a Darwinian mode for 2,000 iterations, optimizing each objective function separately. The resulting objective vectors have been collected in Table 1 for the vehicle crash worthiness problem forming a payoff-table [Miettinen 1999]. In the payoff-table, the ideal point is located on the main diagonal, whereas the nadir point can be approximated by taking the maximum value from each column. Then, from the payoff-table, we can approximate the ideal and nadir points as $\mathbf{z}^*_{\text{crash}} = (1{,}600.0; 6.0; 0.038)$ and $\mathbf{z}^{\text{nadir}}_{\text{crash}} = (1{,}700.0; 12.0; 0.30)$. Notice that from the values shown in Table 1, we have chosen lower values for the ideal point components and higher values for the nadir point components than shown in the table. This was done to assure that the computed objective vectors during the experiments would fall between the ideal and nadir points

in both problems, which is required in the achievement scalarizing function (2). We utilized the payoff-table method to compute the approximations for the ideal and nadir points because of the simplicity of the method, and because our experiments do not critically depend on the exact values of the points—it is sufficient that computed solutions fall between the points, and the points can also be updated during the solution process, if needed.

4.2 Exploring the Effects of the Learning Mode Frequency and the H/L Split on LEMOO Performance

4.2.1 Setup. In this first experiment, our goal was to find out the effect of the choice of the H/L split in the learning mode and the frequency of switching from a Darwinian mode to a learning mode on the performance of our LEMOO method. We chose 10 different H/L splits ranging from 5% to 50% in increments of 5%, and 11 different switching frequencies: 2, 4, 5, 8, 10, 20, 25, 50, 100, 200, and 500. For each combination of an H/L split and switching frequency, we ran our LEMOO method so that the total iterations would always sum to 1,000 (i.e., $N^T \times (N^D + N^L) = 1{,}000$). However, as mentioned in Section 3, we iterate the LEMOO in a learning mode always only once (i.e., $N^L = 1$) throughout the experiment. Each run was repeated 10 times for all problems considered to even out statistical fluctuations arising from the heuristic nature of the method. This meant that our LEMOO method was run 1,100 times in total for each problem considered. As for the reference points chosen in our fitness function (2), we chose $\bar{\mathbf{z}}_{\text{crash}} = (1{,}650.0; 7.0; 0.05)$ for the vehicle crash worthiness problem. The reference point was kept constant throughout the experiments so that the results could be compared.

For each run, we measured and collected the following quantities related to the populations in each of the iterations in Darwinian and learning modes: the decision vectors, the objective vectors, the fitness function values for each individual, the best fitness function value, the average fitness function value, the hypervolume (with the nadir point of the studied problem set as the point with respect to the hypervolume is computed), and the cumulative sum of the number of unique decision vectors. We measured the fitness function values to probe the performance of our LEMOO method in finding optimal solutions. Likewise, we measured the hypervolume and the cumulative sums to probe the variety of solutions in the populations. Since we are utilizing the scalarizing function in (2) to compute the fitness, we are not trying to approximate the whole PO set; instead, the search is focused on finding a small subset of the PO set residing near a reference point. It is therefore feasible to assume the subset of the approximate PO set to form a front of relatively continuous, connected, and smooth objective vectors, which makes the hypervolume and cumulative sums appropriate measures for the variety of the solutions in our case. In addition, solutions close to the reference point are assumed to have approximately a linear connection between the decision variables and objective vectors. The measured and collected quantities were saved for each run in files in a JSON format. Snakemake [Mölder et al. 2021] was used to manage the execution of the experiments and to ensure the reproducibility and transparency of them. The data generated in the experiments is openly available on Zenodo [Misitano 2023b], and the steps needed to reproduce the data are documented in the online documentation of the XLEMOO framework.[4]

4.2.2 Results and Analysis. For each combination of an H/L split and learning mode switching frequency, the data generated by the 10 repetitions of the experiment were aggregated by computing the mean and the standard deviation of each measured quantity. By visual inspection, the data seemed to be normally distributed with no significant outliers, which makes the mean and the standard deviation sensible statistical measures for the data under study.

[4] https://xlemoo.readthedocs.io/en/article_v1.1/introduction.html#reproducibility

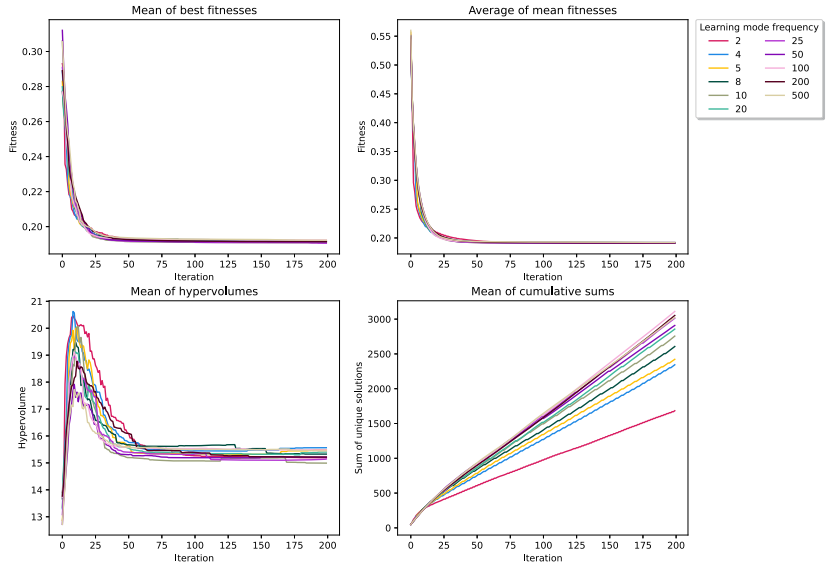

Fig. 3. The aggregated measured data for the vehicle crash worthiness problem after 200 iterations. The data is plotted for all the switching frequencies considered. The H/L split was kept constant at 20.

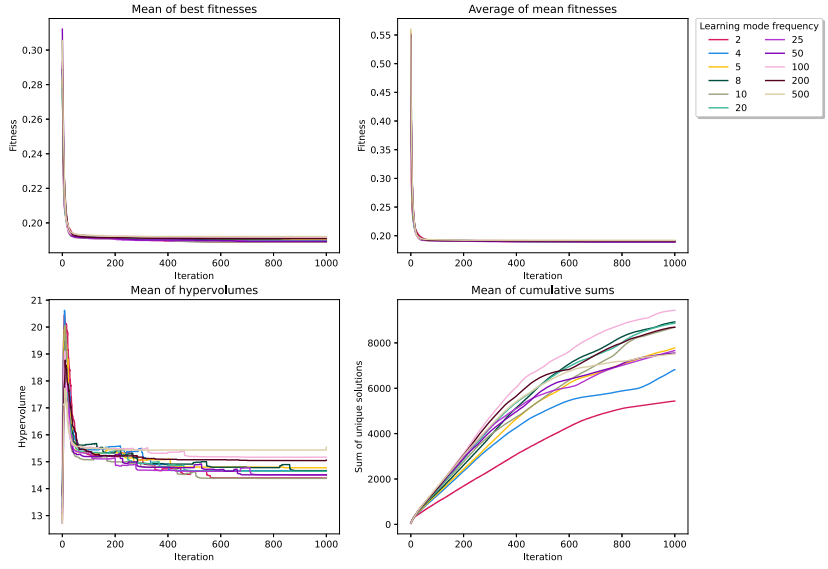

Fig. 4. The aggregated measured data for the vehicle crash worthiness problem after 1,000 iterations. The data is plotted for all the switching frequencies considered. The H/L split was kept constant at 20.

The aggregated data was then further studied by plotting the means of the four measured quantities for each iteration by varying the switching frequency and keeping the H/L split constant. Plots as shown in Figures 3 and 4 were generated and studied for all problems considered. In these figures, the H/L split was set to be 20 to serve as an example. It was quickly noticed that the mean of the best fitness value and the average of the fitness values converged before the 200th iteration in the problems considered, as can be seen by comparing Figures 3 and 4. The mean of the

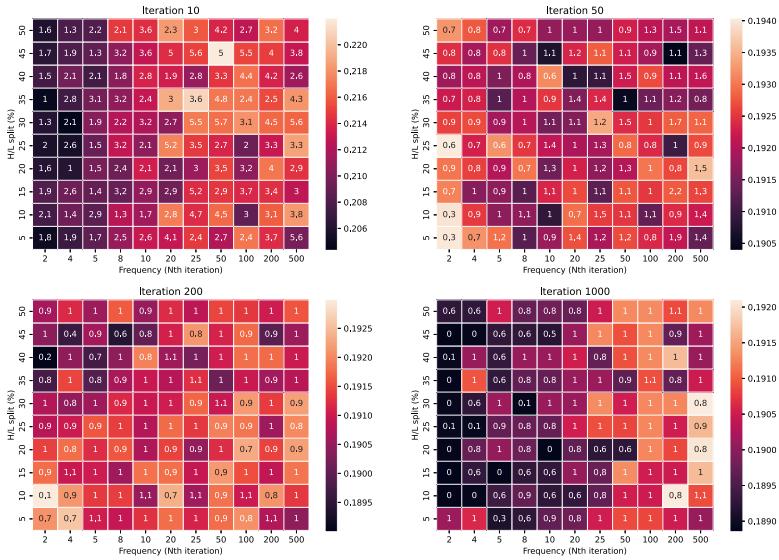

Fig. 5. The mean and standard deviations of the best fitness values for iterations 10, 50, 200, and 1,000 for the vehicle crash worthiness problem.

hypervolume seems to quickly settle before the 200th iteration but clearly keeps decreasing until the last iteration. The mean of the cumulative sums of unique solutions also keeps increasing until the last iteration. The changes after the 200th iteration in all measured quantities seemed approximately constant. The described behavior was true for all H/L splits. Based on these observations, it seems that exploring the early iterations in our LEMOO method in more detail than the later ones makes sense.

As a result of the preceding examination, we decided to study the aggregated data in more detail for iterations 10, 50, 200 and 1,000 by visualizing the data in heatmaps as shown in Figures 5, 6, 7, and 8 for the vehicle crash worthiness problem. In the heatmaps, each cell represents the values of the aggregated measures for an H/L splitting and switching frequency combination. The color of the cell indicates the mean, whereas the numerical quantity inside the cell indicates the standard deviation. The standard deviation is expressed as a percentage of the mean—that is, a quantity's standard deviation of 5, and a mean value of 100, would be expressed as a percentage with a value of 5%. The darker the cell, the lower its values, and the lighter, the higher. For the mean of the best fitness and the average (mean) of the mean fitnesses, a lower value (darker color) is better, whereas for the hypervolume and the cumulative sum of unique solutions, a higher value (lighter color) is better.

4.2.3 Observations. Here we present the results of the experiment and highlight the most obvious observations that can be made based on the data. Our observations are general to all problems considered in the experiment unless specified otherwise. We conclude by giving general remarks about the results. The reader interested only in the final remarks may skip to the end of the observations.

Observing the mean of the best fitness values found in the vehicle crash worthiness problem in Figure 5, we can see that in iteration 10, there is a darker hue on the left side of the heatmap. This means that, on average, the best fitness values found were lower with a lower switching frequency. This trend was also observed in other iterations but was not consistent throughout the problems

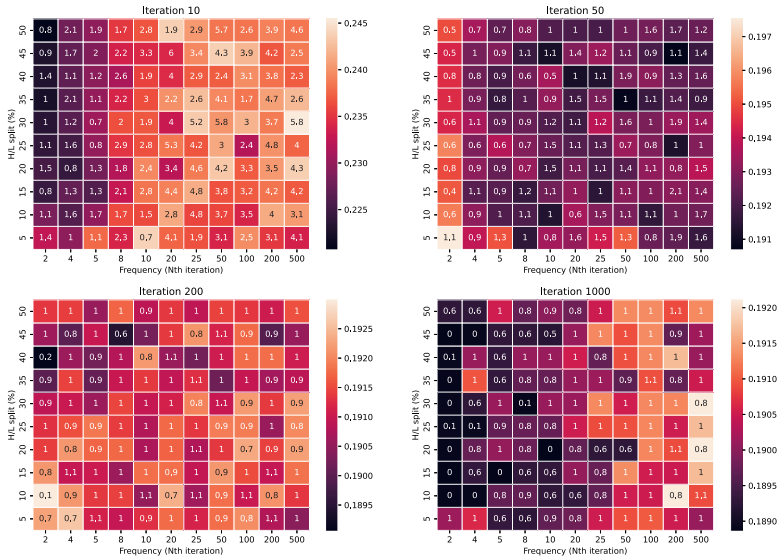

Fig. 6. The mean and standard deviations of the average of the fitness values for iterations 10, 50, 200, and 1,000 for the vehicle crash worthiness problem.

considered—that is, there is a darker hue on the left of the heatmap for iteration 1,000 in the vehicle crash worthiness problem, but this was not generally observed in all problems. We also notice that there is no clear gradient in the vertical directions of any of the heatmaps in Figure 5. This means that the choice of H/L split had no noticeable effect on the performance of our LEMOO method in finding individuals with a better best fitness value.

For the mean of the average fitness values, we notice a similar trend of a darker hue on the left of the heatmap of iteration 10 in Figure 6. A darker hue was again observed on the left of the heatmap for iteration 1,000 for the vehicle crash worthiness problem but was not observed in all problems. We also notice that with a combination of a low switching frequency and H/L split, there might even be a detrimental effect on our method's ability to improve the average of the mean fitness, as seen by the lighter hue of the very left of the heatmap for iteration 50 in Figure 6. We again observe no noticeable change in gradient in the vertical direction in any of the heatmaps, which means that the H/L split had little effect on our method's performance to improve the populations' average means values.

Observing the mean values for the computed hypervolumes in the vehicle crash worthiness problem show in Figure 7, we can in turn note that in the early iterations (10 and 50), there seems to be a lighter hue on the left indicting higher (better) values for the mean value of the hypervolume computed. For the vehicle crash worthiness problem, there seems to be a slightly darker hue in the heatmap of iteration 1,000, indicating lower (worse) hypervolume values, but this was not observed in all problems. And again, the choice of H/L split seems to have had little effect on the mean values of the hypervolume since there is no noticeable gradient in the vertical direction of any of the heatmaps.

If we study the mean values of the cumulative sums of unique solutions found in each iteration for the vehicle crash worthiness problem shown in Figure 8, it seems that in early iterations (10 and 50), with a combination of a low switching frequency and H/L split, our method seems to perform better when finding unique solutions. However, in the other iterations, there seems to be a lighter hue on the right of the heatmaps, indicating a higher (better) value for the cumulative

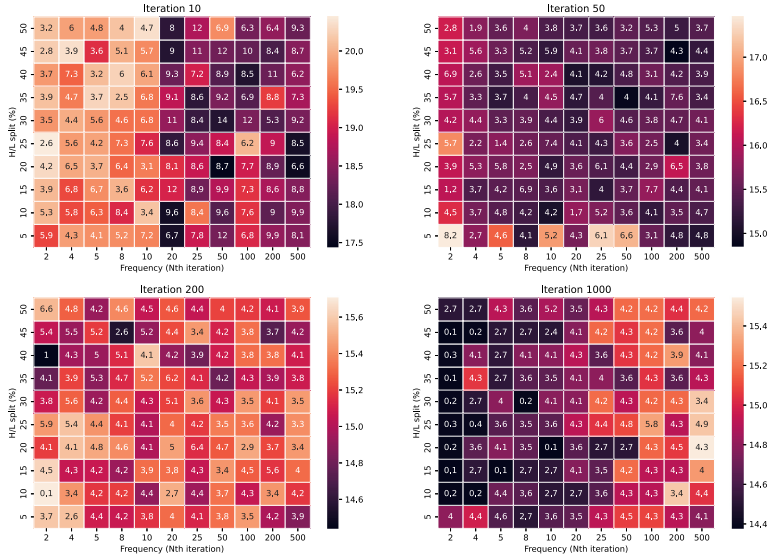

Fig. 7. The mean and standard deviations of the hypervolumes values of the population in iterations 10, 50, 200, and 1,000 for the vehicle crash worthiness problem.

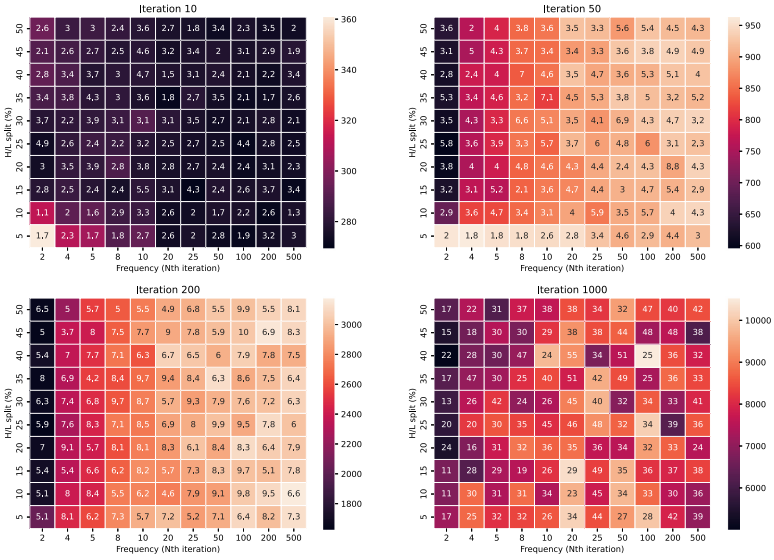

Fig. 8. The mean and standard deviations of the cumulative sums of unique solutions in iterations 10, 50, 200, and 1,000 for the vehicle crash worthiness problem.

sums. Unlike in the observations for the other measures, it seems that the choice of the H/L split might have a minor effect on our method's ability to find more unique solutions.

Last, observing the standard deviations of the measured quantities in the heatmaps in Figures 5 through 8, we can observe mostly low values between 0% to 10%, indicating a relatively low statistical variance of our results. However, an exception to this observation can be seen in iteration 1,000 for the cumulative sums of unique solutions in Figure 8, where the standard deviations raise

occasionally above 20%, and in some cases even above 50%. Nevertheless, in general, the statistical fluctuations of our results were low.

Final Remarks. For each of the measured quantities (the best fitness value, the average of the mean fitness value, the hypervolume, and the cumulative sum of unique solutions), the most noticeable effects were observed in early iterations (10 and 50). The effects were positive. In all of the heatmaps, the effect were noticed only when the switching frequency was lower than the iteration represented by the heatmap—for example, in the heatmaps representing results for iteration 50, the noticed effects were present only for switching frequencies lower than 50. For higher switching frequencies in these cases, our LEMOO method has not engaged in a learning mode yet, meaning that the method has operated purely in a Darwinian mode. This is an important observation, as it is clear evidence that the addition of a learning mode has had an obvious effect on the performance of our LEMOO method.

4.3 Explanations in LEMOO

Here, we explore in more depth the explanations emerging from the use of an interpretable ML model in the learning mode of our LEMOO method. We also discuss how rule sets to support DMs can be extracted from our method. These rule sets provide insights about the decision vectors with the highest fitnesses found. We focus on the vehicle crash worthiness problem to give a concrete example of the explanations generated. A showcase on how the rules can benefit a DM in practice is given in Section 5.

4.3.1 Setup and Results. Here, we utilized our LEMOO method described in the previous sections. We chose the number of total iterations to be 200 and the switching frequency to be 20. The H/L split value was set to 20%. These choices were based on the results of the experiments conducted in Section 4.2. The other parameters of our LEMOO method were kept the same as described previously. We ran the method once with four different reference points used in the fitness function (2), which are listed in Table 2. The first reference point $\bar{\mathbf{z}}_1$ was the same as used in the previous experiments. There was no particular criterion in choosing the other reference points other than for the sake of providing more examples of possible rule sets emerging from our method to study.

Each rule in the rule sets was generated as follows. Rules were extracted from the skope-rules model from the final iteration of our LEMOO method. Each rule was then inspected, and the most accurate rule was selected. From the most accurate rules, upper and lower limits for each variable in the vehicle crash worthiness problem were searched for. If there was no rule describing a variable's lower or upper limit, the limit was taken from the final population computed in the LEMOO method—that is, the population was inspected for the variable, and the lowest or highest value found in the population was used as the lower or higher limit, respectively. If two rules extracted from the skope-rules had the same accuracy and described the same variable, then the rule with a stricter limit was selected—that is, if two lower (higher) limits were extracted from the rules, then the lower (higher) limit with a higher (lower) value was selected.

The rule sets generated are shown in Table 2 for each reference point considered. The accuracies of the rules from which the limits were extracted are also shown. An accuracy of −1 means that no rule described a particular variable's lower or higher limit, so the limit extracted from the population was used instead. In Table 2, we also report the lower and upper limits extracted from the final population to be compared with the respective limits extracted utilizing the rules from skope-rules.

To study the spread of the possible solutions computed based on the rules in the rule sets generated, we randomly generated 100,000 decision vectors based on the rules—that is, for each variable,

Table 2. Rule Sets Generated for the Four Reference Points Considered

Rule set 1: $\bar{\mathbf{z}}_1 = (1{,}650.0; 7.0; 0.05)$						
Variable	Lower (R)	Acc.	Upper (R)	Acc.	Lower (P)	Upper (P)
x_1	1.0	(0.485)	1.0	(1.0)	1.0	1.0
x_2	1.37958	(1.0)	1.43105	(1.0)	1.42731	1.42731
x_3	1.0	(0.468)	1.00002	(1.0)	1.0	1.0
x_4	1.0	(0.499)	1.00197	(1.0)	1.0	1.0
x_5	2.31826	(1.0)	2.58524	(1.0)	2.38275	2.38275
Rule set 2: $\bar{\mathbf{z}}_2 = (1{,}600.0; 8.0; 0.07)$						
Variable	Lower (R)	Acc.	Upper (R)	Acc.	Lower (P)	Upper (P)
x_1	1.0	(0.518)	1.00513	(1.0)	1.0	1.0
x_2	1.0	(0.478)	1.0	(1.0)	1.0	1.0
x_3	1.0	(0.404)	1.00004	(1.0)	1.0	1.0
x_4	1.0	(-1)	1.00428	(1.0)	1.0	1.0
x_5	1.0	(0.232)	1.0	(1.0)	1.0	1.0
Rule set 3: $\bar{\mathbf{z}}_3 = (1{,}700.0; 6.5; 0.18)$						
Variable	Lower (R)	Acc.	Upper (R)	Acc.	Lower (P)	Upper (P)
x_1	1.0	(-1)	1.00001	(1.0)	1.0	1.0
x_2	2.99992	(0.997)	3.0	(0.496)	3.0	3.0
x_3	2.56003	(1.0)	2.57444	(1.0)	2.56865	2.56929
x_4	1.0	(0.487)	1.00001	(1.0)	1.0	1.0
x_5	2.97299	(1.0)	2.99937	(-1)	2.99717	2.99937
Rule set 4: $\bar{\mathbf{z}}_4 = (1{,}695.0; 6.1; 0.04)$						
Variable	Lower (R)	Acc.	Upper (R)	Acc.	Lower (P)	Upper (P)
x_1	1.00001	(1.0)	1.00165	(1.0)	1.00001	1.00001
x_2	2.99559	(1.0)	3.0	(0.987)	3.0	3.0
x_3	1.0	(-1)	1.00001	(1.0)	1.0	1.0
x_4	1.08398	(1.0)	1.1352	(1.0)	1.10772	1.10772
x_5	2.99449	(1.0)	3.0	(-1)	3.0	3.0

The lower and upper limits extracted from the skope-rules and final population are listed in columns 'Lower (R)' and 'Upper (R),' respectively, whereas the limits extracted from the final population alone are listed in columns 'Lower (P)' and 'Upper (P),' respectively. The limits extracted from a skope-rules are followed by the respective rule's accuracy listed in the 'Acc.' columns, where 1 is the highest possible accuracy (equal to 100%) and 0 is the lowest. If a limit was extracted from the final population instead, then the reported accuracy is -1.

we generated a random value between the lower and upper limits reported in the rule describing the variable. We then plotted a histogram for each set of randomly generated decision vectors showing the distribution of the vector's fitness value. These histograms are shown in Figure 9.

The process of running our LEMOO method and generating the described rule sets can be found in a Jupyter notebook.[5] This notebook serves also as a good example of how our LEMOO method can be, and has been, implemented in practice.

4.3.2 Observations. Observing the rule sets in Table 2, we can make some observations. First, the rules extracted from skope-rules always describe a wider, or equal, range of possible decision

[5] https://xlemoo.readthedocs.io/en/article_v1.1/notebooks/How_to_extract_rules_example.html

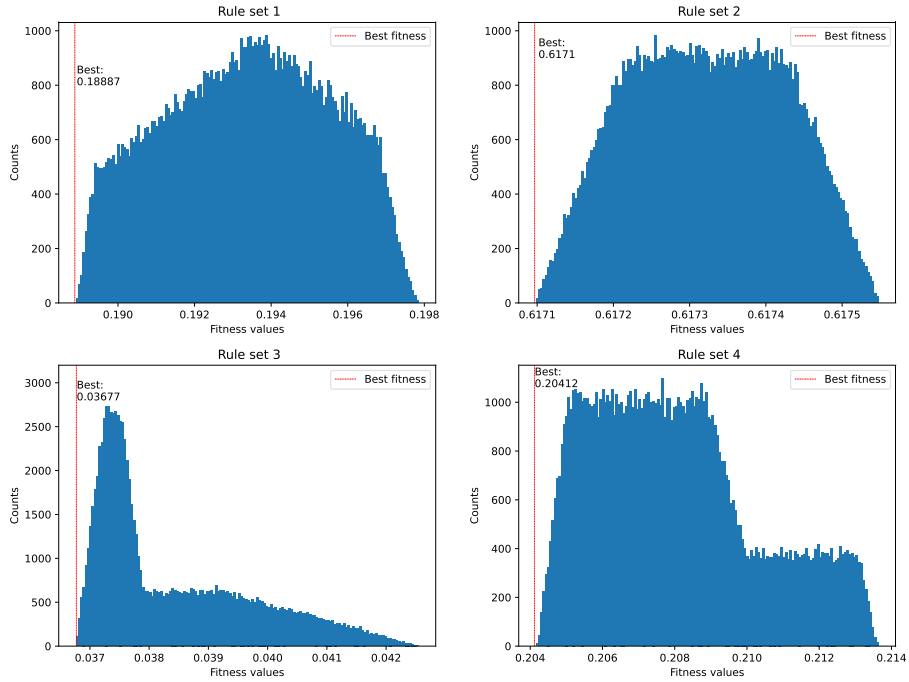

Fig. 9. The distributions of the fitness values of the decision vectors generated based on the rule sets extracted for the car-side impact problem. The best fitness function value found by our LEMOO method for each reference point is also shown in the plots. The number of samples in each histogram is 100,000.

variable values when compared to the rules extracted from the final population alone. Second, when the range is the same, or when there is no range—the upper and lower limits are the same—the accuracies of the upper or lower limits extracted from skope-rules are in the range $[0, 1)$. And third, in most cases, at least one rule extracted from skope-rules was found to describe the lower and upper limits of a variable.

Inspecting the histograms in Figure 9, none of the rule sets describe solutions that have fitness values better than the best fitness value found. This is expected since after 200 iterations, our LEMOO method was observed in Section 4.2 to have already converged in terms of best fitness value and the average of fitness values in a population. Nonetheless, the rules in the rule sets describe solutions with fitness values relatively close to the best. If we assume for the objective vector corresponding to the individual with the best fitness value to be close to the reference point considered when generating each rule set, we can also assume that the objective vectors corresponding to the decision vectors generated based on the rule sets to be close to the reference point. In the histograms for rule sets 1 and 2 in Figure 9, we can see an even distribution for the solution fitnesses, whereas in the histograms for rule sets 3 and 4, we can see a slight bias in the fitness values toward the best fitness value. Out of $100,000$ solutions generated based on the rules in each rule set, none of the fitness values are significantly far from the best fitness values. This means that by varying the decision vectors according to the rules in the rule sets, we find solutions with respective objective vectors in the proximity of the objective vector with the decision vector corresponding to the best fitness value. This also means that near the reference points considered, we can approximate the relationship between objective function values and decision variables to be linear.

Final Remarks. The rule sets extracted from the interpretable ML model in our LEMOO method are clearly more informative about the possible ranges of decision variable values close to a solution of interest from the perspective of a DM when compared to the respective information extracted from the final population alone. We also saw that the solutions generated according to the rules in the rule sets are close to the original best solution. This can help a DM explore alternative solutions close to the best one. Since the rule sets were learned based on past populations as well, and not just the final one, we conclude that a better insight about the connection between decision vectors and objective vectors close to a solution of interest can be derived by considering the population history, and that utilizing interpretable ML is one possible approach to generate these insights.

5 SHOWCASE OF THE ADDED BENEFITS OF EXPLAINABILITY

In this section, we demonstrate in a proof of concept the benefits of the explanations extracted from our LEMOO method, and how they can support a DM in solving a MOO problem. More technical details on how the results have been computed and the rules extracted were given in Section 4, where we reported more detailed experiments to explore the performance and explanations provided by our LEMOO method, and presented the implementation of our method in more detail.

5.1 LEMOO as an Explainable Interactive Method

We demonstrate how our LEMOO method, which combines an indicator-based EMO and skope-rules, can be used as a reference point based interactive MOO method. The input is a reference point provided by a DM. The reference point is utilized in the scalarizing function in (2), which functions as the indicator in our method. Our method then computes a population of solutions from which the solution with the best fitness value is shown to the DM. In addition, the method outputs explanations, in the form of rule sets generated by skope-rules, describing the high-performing solutions providing the DM with additional insights about the best-performing solutions generated near the reference point from the perspective of decision variables. This idea is visualized in Figure 10. In addition, the DM has also the option to provide decision vectors, which are evaluated into objective vectors, to explore different solutions inspired by the insights provided by the explanations. This in turn can also provide support to the DM in providing further reference points in subsequent iterations. In practice, an *analyst* operating the method can aid the DM when evaluating decision vectors specified by them. But in our case, we have assumed the DM to be an engineer, who is familiar with the domain of the problem and its technicalities, including the meaning of the decision vectors.

As a showcase of how our method can be utilized as an interactive MOO method, we show three iterations with the author acting as the DM. We also comment on the insights provided to the DM. We consider the *vehicle crash worthiness problem* [Liao et al. 2008] to optimize the crash safety of a car. It has three objectives to be minimized: the frontal mass of the vehicle, the collision acceleration experienced by passengers in case of a full frontal crash, and the toe board intrusion in the case of an off-set frontal crash. The problem has five decision variables, which model the thickness of five reinforced members in the frontal structure of the car. They have box-constraints, where the thickness should be between 1 and 3 mm. The problem has no other constraints.

To support the DM in providing reference points, we assume that before the optimization process, the DM has been informed of the approximated ideal and nadir points of the vehicle crash worthiness problem: $\mathbf{z}^*_{\text{crash}} = (1{,}600.0; 6.0; 0.038)$ and $\mathbf{z}^{\text{nadir}}_{\text{crash}} = (1{,}700.0; 12.0; 0.30)$. Details on how these points were approximated are given in Section 4. The explanations provided to the DM during the optimization process are the rule sets shown in Table 3. These describe the solutions with

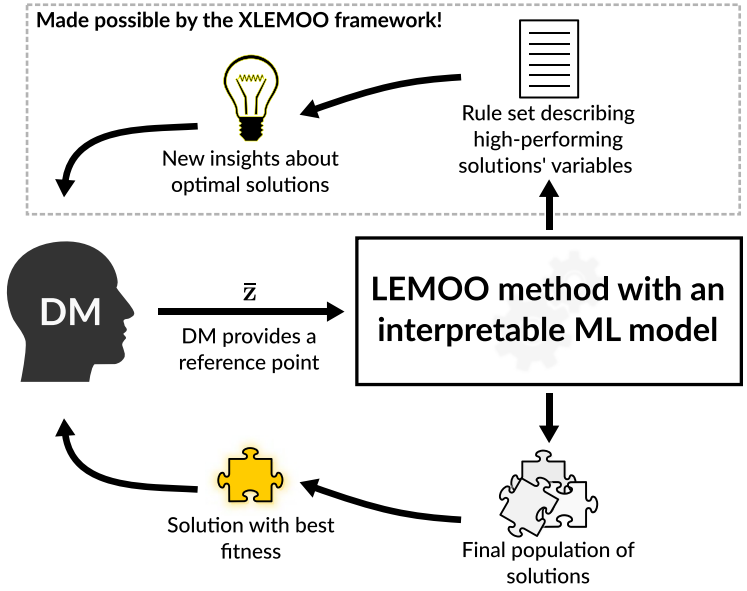

Fig. 10. An example of how our LEMOO method can be used as an interactive MOO method.

the best fitness values found in each iteration by means of lower and upper limits for each decision variable. Each limit has also an accuracy associated with it. The rule sets are extracted from the skope-rules model used in our method. A more detailed description of how the rule sets have been derived is given in Section 4.3. Next, we describe the three iterations and show how our LEMOO method can support a DM in solving the vehicle crash worthiness problem and gaining new insights.

Iteration 1. In the first iteration, the DM provided an optimistic reference point close to the ideal point of the problem: $\bar{\mathbf{z}}_1 = (1{,}610.0; 6.2; 0.041)$, which resulted in the best solution (as defined earlier) $\mathbf{x}_1 = (1.00887; 3.0; 1.0; 1.37784; 1.00168)$ with the objective vector $\mathbf{z}_1 = (1{,}669.28; 7.69556; 0.08126)$. The DM was also shown the rule set derived shown in *Iteration 1* in Table 3. The DM noticed that the objective vector was worse than the reference point in each component. Looking at the rule set in Table 3 for *Iteration 1*, he noticed that the thickness of the second and fourth member could perhaps be lowered while keeping the result otherwise similar to decrease the mass (first objective) by a small amount. Driven by these insights about the optimal solutions, he then tried evaluating the problem with the decision vector $\mathbf{x}'_1 = (1.00887; 2.99985; 1.0; 1.36258; 1.00168)$, which resulted in the objective vector $\mathbf{z}'_1 = (1{,}669.18; 7.66459; 0.08166)$. Although the effect on the mass was minimal, he noticed a more significant improvement in the collision acceleration experienced by passengers (second objective). After learning of the minimal effect changing the variables had on the mass, and the too optimistic value he had set for the mass in the aspiration level of $\bar{\mathbf{z}}_1$, he decided to explore the mass by providing a new reference point instead.

Iteration 2. In the next reference point, the DM decided to increase the aspirations levels for the mass, keep the collision acceleration the same as in $\mathbf{z}'_1$, but increase the aspiration level for the toe board intrusion (third objective) in hopes of achieving a lower mass than in the previous iteration. Thus, he gave the reference point $\bar{\mathbf{z}}_2 = (1{,}680.0; 7.66459; 0.07)$, which resulted in the best solution $\mathbf{x}_2 = (1.0; 3.0; 1.0; 1.25473; 2.99988)$ with the corresponding objective vector $\mathbf{z}_2 = (1{,}677.23; 7.61449; 0.068647)$. The DM noticed that the objective values in the objective vector were all better than in the reference point, but the mass was now clearly too high. After inspecting the rules in the rule set in Table 3 for *Iteration 2*, he noticed that the fourth variable once again had

a large range. The DM made a note of this and decided not to try other decision variable values based on the rules in the rule set because the ranges for the variables were narrow and did not encourage the DM to explore the solution in this iteration any further.

Iteration 3. The DM was satisfied with the value of acceleration experienced by passengers (second objective) in the previous objective vector, $\mathbf{z}_2$; however, he still wished to find a solution with a lower mass (first objective) and was ready to trade off for a slightly larger value for the toe board intrusion (third objective). Thus, the third reference point was $\bar{\mathbf{z}}_3 = (1{,}670.0; 7.61449; 0.085)$, which led to the best solution $\mathbf{x}_3 = (1.00286; 2.99999; 1.00000; 1.30009; 1.03374)$ with the objective vector $\mathbf{z}_3 = (1{,}668.83; 7.53771; 0.083098)$. The DM was happy to see a lower mass (first objective) and a lower acceleration (second objective) than in $\mathbf{z}_2$. After inspecting the rule set of *Iteration 3* in Table 3, the DM noticed again that the fourth variable of the problem had the largest range of

Table 3. Three Iterations Featured in Our LEMOO Method's Showcase

Iteration 1				
Reference point: $\bar{\mathbf{z}}_1 = (1{,}610.0; 6.2; 0.041)$				
Best objective vector: $\mathbf{z}_1 = (1{,}669.28; 7.69556; 0.08126)$				
Best decision vector: $\mathbf{x}_1 = (1.00887; 3.0; 1.0; 1.37784; 1.00168)$				
Variable	Lower	Acc.	Upper	Acc.
x_1	1.00887	(−1)	1.00973	(1.0)
x_2	2.99985	(1.0)	3.0	(−1)
x_3	1.0	(0.496)	1.0	(0.429)
x_4	1.36258	(1.0)	1.37986	(1.0)
x_5	1.00168	(−1)	1.0028	(1.0)
Iteration 2				
Reference point: $\bar{\mathbf{z}}_2 = (1{,}680.0; 7.66459; 0.07)$				
Best objective vector: $\mathbf{z}_2 = (1{,}677.23; 7.61449; 0.068647)$				
Best decision vector: $\mathbf{x}_2 = (1.0; 3.0; 1.0; 1.25473; 2.99988)$				
Variable	Lower	Acc.	Upper	Acc.
x_1	1.0	(−1)	1.00002	(1.0)
x_2	2.99979	(1.0)	3.0	(0.496)
x_3	1.0	(0.993)	1.0	(1.0)
x_4	1.23885	(1.0)	1.25545	(1.0)
x_5	2.99986	(1.0)	2.99988	(0.997)
Iteration 3				
Reference point: $\bar{\mathbf{z}}_3 = (1{,}670.0; 7.61449; 0.085)$				
Best objective vector: $\mathbf{z}_3 = (1{,}668.83; 7.53771; 0.083098)$				
Best decision vector: $\mathbf{x}_3 = (1.00286; 2.99999; 1.00000; 1.30009; 1.03374)$				
Variable	Lower	Acc.	Upper	Acc.
x_1	1.00184	(0.996)	1.00558	(1.0)
x_2	2.7135	(1.0)	3.0	(0.48)
x_3	1.0	(0.517)	1.00011	(1.0)
x_4	1.21741	(1.0)	2.12628	(1.0)
x_5	1.02602	(1.0)	1.04323	(1.0)

The reference point provided by the DM, the decision and objective vector corresponding to the best solution, and the rules derived from our method are shown for each iteration. The lower limits are shown in the 'Lower' column and the upper limits in the 'Upper' column. Each limit is followed by its accuracy ranging from 0 (completely inaccurate) to 1 (completely accurate). Refer to Section 4.3 for further details on how the rule sets have been generated.

values according to the rules—this time noticeably larger than in the previous iterations. Although the DM was already happy with the solution found, the insights from the rule set inspired him to modify the decision vector according to the rules in hopes to decrease the mass by lowering the thickness of the members represented by the decision variables x_1, x_2, x_4, and x_5. This led to the modified decision vector $\mathbf{x_3}' = (1.00184; 2.7135; 1.00000; 1.21741; 1.02602)$, with the corresponding objective vector $\mathbf{z_3}' = (1{,}667.49, 7.54346, 0.094955)$. The DM was quick to note that lowering the thicknesses did not have a significant effect on the mass (first objective) or acceleration (second objective), but it did noticeably increase the toe board intrusion (third objective), which was too large a tradeoff for the DM to make.

In the end, the DM chose $\mathbf{x_3}$ as the final solution since he was most satisfied with the corresponding objective values. However, the DM also learned that changing the thicknesses of the five reinforced members in the frontal structure of the car represented by the decision variables had, in general, a surprisingly small effect on the mass of the car and a more noticeable effect on the toe board intrusion. Furthermore, the DM observed that the rule sets include larger ranges for variable x_4, as indicated in Table 3. The DM thought that this might diminish the impact of the variable on the values of the objective function, particularly within his region of interest. The DM thought that it may be a good idea to revisit the problem formulation to investigate this last point further before making any decisions based on the final solution.

Final Remarks on the Showcase. As we saw, thanks to the explanations (rule sets), the DM got important insights regarding the connections between decision variables and objective vectors, and he got some support to provide new reference points. Moreover, the insights made the DM question the formulation of the MOO problem, since changing the variables did not have the effect he expected. This caused the DM to postpone any decision based on the final solution found. The insights gained from the explanations (rule sets in Table 3) caused the DM to be more critical about the solutions and also explore the decision space of the problem, which, in the end, affected the whole decision making process.

6 DISCUSSION

In this section, we discuss the results of the experiments conducted in Section 4 and the results' main implications in regard to our current work and future research. We also consider the implications of the practical showcase discussed in Section 5. We discuss the results of the experiments related to the performance of our LEMOO method in Section 6.1, and the results regarding the explanation aspects of our method and the showcase in Section 6.2. We then discuss the further potential of the explanations in the context of LEMOO in Section 6.3. Finally, we conclude by discussing the general implications of our work in Section 6.4.

6.1 On the Performance of LEMOO

Based on the observations made in regard to the performance of our LEMOO method in Section 4.2, we can draw the following conclusions. First, a learning mode has a clear positive contribution to the performance of our LEMOO method in finding individuals with better fitness values if the frequency of switching to the learning mode is not too frequent nor too infrequent. This can help a LEMOO method converge faster if compared to a method without a learning mode. However, if the switching is too frequent, the population might not include enough individuals with clearly better fitness values, which makes it difficult for the ML model in the learning mode to be able to build a valid hypothesis for the high-performing group. In turn, if the learning frequency is too infrequent, the populations can become too homogeneous, which again makes it hard for an ML model to learn a distinction between a high-performing and low-performing individual. Interestingly, the choice of an H/L split seems to have little to no effect on the performance of our LEMOO method unless

combined with a low switching frequency, in which case it can have a detrimental effect on the performance. The fact that LEMOO offers clear, although not immense, benefits in term of search performance of finding solutions with a better fitness value is in line with the findings reported in previous studies exploring LEMOO methods, which were discussed in Section 2.3. This gives us confidence in the validity of our own experiment as well. But in these previous works, the effect of the switching frequency or H/L split was not studied in detail.

The second conclusion we can make is that a low switching frequency can boost the diversity of the populations found in our LEMOO method, but only during early iterations (up to 200 out of 1,000 in our case). The observations for the computed hypervolumes and cumulative sums in Section 4.2 are the evidence supporting this claim. This makes sense if we take into account the first conclusion we made. In other words, because with a lower switching frequency, our LEMOO method converges faster, and it also means that the populations become saturated with more similar individuals in earlier iterations, leading to a decrease of the learning mode's performance in the hypothesis forming. We did, however, observe that a low H/L split combined with a low switching frequency can lead to a slight boost in the hypervolumes and cumulative sums of the populations, but if we also take into the account the observation that this same combination of frequency and split can have a detrimental effect on the fitnesses of the individuals found, the tradeoff between finding good fitness values and a diverse population with a low frequency and split becomes evident. The potential of LEMOO boosting the diversity of computed populations was also observed in the works discussed in Section 2.3; however, the potentially detrimental effect of a low switching frequency was neither reported nor studied, or both, in these previous works.

It is important to also mention the limitations of our first experiment. First, we only considered two MOO problems in Section 4.2, we did not vary any parameters of the Darwinian mode of our method, we did not vary the ML model used in the learning mode of our method, we did not consider other fitness functions than the scalarizing function in (2), and we did not vary the parameters of the ML model used. We did, however, explore these aspect during internal testing. We chose to report the results for only two problems because we found no significant differences in the results for other problems tested. This was also true for the fitness function chosen—we tried other scalarizing functions as well with no significant changes in the performance of our LEMOO method. Not surprisingly, the parameters in the Darwinian mode of our method did have a noticeable effect on the performance, but the results showed still similar trends when compared to what we reported in this study. The choice of our ML model to be skope-rules was mainly motivated by its superior performance in finding rules. We compared it to a decision tree (C4.5 [Salzberg 1994] and CART [Breiman 2017]), boosted rule set [Freund and Schapire 1997], and slippery rule set [Cohen and Singer 1999] but simply found their performance subpar compared to skope-rules. One of the main factors in choosing an ML model, apart from its interpretable nature, is its performance. It must be fast enough so that training it and generating new solutions is comparable to the time it takes for the LEMOO method to finish iterating in a Darwinian mode. Otherwise, the learning mode of a LEMOO method might be too slow to justify its inclusion. In our setting, we found skope-rules to be able to find useful rules in a feasible time, which ultimately led us to choose it as our ML method. The issue of exploring other ML models is also technical since the trained models are used to generate new data, which is an atypical application of the models. This means that it requires extra effort to develop the necessary utilities for the models to be utilized in a LEMOO method. In our XLEMOO framework, we have provided the necessary utilities to use the aforementioned ML models. That being said, our goal was to explore the effect of the H/L split and the switching frequency to a learning mode in our LEMOO method.

We have not compared our numerical results to previous works. This is because similar works with comparable results and reproducible experiments do not exist according to our best

knowledge. However, we have taken care to report our results in a way that is reproducible so that future works may be compared to ours. Because we propose our current method to be used as an interactive MOO method, challenges arise since comparing interactive methods to each other is an open research question [Afsar et al. 2021], which is further complicated by the inclusion of explanations. The best works to compare our method to would be the ones presented in the work of Corrente et al. [2021] and Misitano et al. [2022], since they both discuss an interactive MOO method with explanations. However, there is no established way to compare such methods. To best compare these methods, we suggest that in a future study, experiments with human participants should be conducted to compare the interactive methods (as suggested by Afsar et al. [2022]), where the usefulness of the explanations to a human user is also assessed. That being said, our current results can be used as a baseline when developing new and more advanced LEMOO methods, and as an initial guideline when figuring out the H/L splitting ratios and switching frequencies, for instance.

6.2 On the Explanation Aspects of LEMOO

In Sections 4.3 and 5, we studied the potential of the generated explanations in our LEMOO method when the ML model employed in a learning mode is interpretable. In light of the showcase and the observations made, we conclude the following. First, it is evident that the rules extracted from the skope-rules provide much more potential insight for a DM about the decision variable values near a reference point if compared to the rules extracted from the final population alone. It was also observed that certain decision variables were clearly highlighted as being important when different reference points were used. This importance was underlined by the fact that rules defining a sensible—as in not being too narrow—range were found only for a couple of variables in each rule set. This can help a DM learn about the MOO problem by helping them build an understanding of which decision variables could be varied in the final solution while not steering too far from a point of interest (i.e., the reference point). This can guide the DM in providing further preferences, or in making a final decision based on the solutions found, as we saw in the showcase.

In addition, the accuracies of the rules in each rule set observed can prove to be useful for a DM. When a rule defines a very narrow range, or no range at all, if the accuracy of the lower or upper limit is low, it can encourage the DM to explore solutions beyond these more inaccurate ranges. This can help the DM explore regions in the decision space of the MOO problem that could otherwise be left unexplored but still potentially contain solutions of interest.

We believe that our LEMOO method can be best utilized as an interactive method, as was done in Section 5.1, where a DM provides multiple reference points in subsequent runs of our method. This way, the explanations may be exploited to their fullest by the DM helping them learn about the MOO problem and explore the solution space.

Our experiments in Section 4.3 were of course subject to the same limitations as discussed in Section 6.2. Furthermore, to better validate the actual usefulness of the rule sets generated based on our LEMOO method in an interactive setting, experiments with real DMs should be conducted. Although rule sets are, in general, easily interpretable by humans [van der Waa et al. 2021], it would be interesting to compare different kinds of explanations in an interactive MOO context. The way explanations are communicated to the DM in a MOO is also an unexplored area (e.g., how to best visualize them?). Empirical experiments, such as the one presented by Afsar et al. [2022], could be adapted to explore how explanations, and their usefulness, are perceived by DMs.

6.3 Further Potential of Explainability in LEMOO

In the showcase given in Section 5, we saw how the explainability provided by our LEMOO method was able to offer support to a DM in gaining insights of the connection between decision

variables and objective vectors in providing new reference points, and in assessing the formulation of the MOO problem. For providing new reference points, it is also possible to present the high-performing individuals in the final population of our method to the DM in the objective space of the problem, effectively providing the DM with not only further solution candidates but also a description of the candidates in the objective space. Showing the high-performing solution candidates in the objective space can support a DM in providing further reference points as well, which addresses the open issue in interactive MOO methods—that is, the lack of supporting DMs in providing preferences during an interactive solutions process.

Although a more technical DM may appreciate the implications of explainability regarding a MOO problem formulation, the explanations can also offer valuable information to an analyst formulating the MOO problem. An analyst can explore a formulated problem by trying to generate an approximation of its whole PO set of solutions instead of just a subset near a reference point, as we have done in our current work. A whole set can be approximated by choosing the indicator appropriately in an indicator-based method, or the EMO method in the LEMOO approach can be switched to some other method altogether. Nonetheless, assuming an indicator-based method with rule-based explanations, an analyst can gain important insights about the nature of the whole approximated PO set. For instance, the rules may reveal that in the computed approximation, the ranges of the decision variables may differ significantly from the box-constraints of the problem. In turn, this can cause the analyst to revisit the MOO problem formulation by adjusting the constraints on the variables leading to a computationally more efficient problem formulation (achieved by limiting the search space). Rules significantly differing from the box-constraints of the MOO problem may also indicate that a search process for optimal solutions has been stuck on a certain part of the PO set of solutions—a local minima, for instance—which can be the case with discontinuous PO solution sets. Thus, explanations can enhance the process of formulating and validating MOO problems and assessing the effectiveness of the optimization methods chosen, for example, when modeling MOO problems in collaboration between domain experts and analysts as described in the work of Afsar et al. [2023].

In the example given in Section 5.1, we saw how a single solution could be accompanied by descriptive rules describing similar solutions near it in terms of decision variables. This can also facilitate communicating the found solution to other stakeholders, improving the transparency of the decision making process. This can help DMs in justifying their decisions to not just stakeholders but to other DMs as well, such as in a group decision making process [Lu et al. 2007]. For instance, explanations can offer negotiation support between two or multiple DMs with different preferences. It is possible that different objective vectors may share similarities in the decision space, which can be unveiled by the kind of rule sets discussed in this section. This can support multiple DMs in finding a compromise solution by considering decision variable values.

Thus far, we have considered the usefulness of explanations to a DM or analyst based on the final population and the history of populations preceding it. As we saw in Section 4, the ML model used, in our case skope-rules, was able to boost the search process for optimal solutions during the optimization process based on intermediate populations, and we may also extract the explanations generated on these intermediate populations during the optimization process to provide important insights to a DM. Following the NAUTILUS philosophy[6] [Miettinen and Ruiz 2016] and applying it to the interactive method discussed in Section 5.1, we can choose to iterate the LEMOO method only a few times and produce a suboptimal population of solutions, and show the high-performing solutions of this population to the DM. This allows the DM to provide new preferences, which can then be used in the indicator (in our case, the achievement scalarizing function in (2)) to steer the

[6]Starting an interactive process from the nadir point of a MOO problem and iteratively improving upon it until a solution on the PO set is reached.

optimization process toward solutions of interest from the point of view of the DM before actual convergence is reached in the LEMOO method. This can promote the exploratory aspects of the LEMOO method when employed as an interactive MOO method, and enables the DM to make a tradeoff free decision during the interactive process. Unlike existing NAUTILUS methods, applying the same philosophy to a LEMOO approach also provides information on the intermediate decision variable values, which are not available in NAUTILUS methods. This can make the comparison of intermediate solutions more meaningful from the perspective of the DM.

Although we have not explored the ideas discussed in this subsection explicitly in our current work, the XLEMOO framework described in Section 3.3 enables researchers and practitioners to readily start exploring the discussed ideas and more. Our current work provides a proof of concept on the potential of explainability in the context of LEMOO promoting the overall paradigm of explainable MOO.

6.4 General Implications of Our Study and Moving Forward

It is perhaps a little surprising that LEMs have not been explored more in the context of EMO methods given the potential shown by our and past studies of LEMOO methods. The fact that a lot of individuals are generated in the many populations of an EMO method makes the application of ML models, which usually need significant amounts data to perform well, not only possible but perchance even natural.

Based on our study, a simple addition of a learning mode to an indicator-based EMO method could work as a niching operator to increase the diversity of individuals, or to boost the search performance of the method when searching for individuals with better fitness values. By taking into account the whole population history, a learning mode could also be a way to implement a global search step in an EMO method. Moreover, as long as the populations generated in an EMO method are saved in an archive, the same procedure to generate rule sets described in Section 4.3 could be applied to any EMO method, making the method more explainable. Of course, mixing EMO methods with ML does come with an increase in computation cost, but as we have shown, even simpler interpretable ML models, such as skope-rules, can work when utilized in a learning mode.

Going forward, it would be interesting to see the concept of LEMs explored in other types of EMO methods, such as in dominance-based and decomposition-based methods, and exploit their unique nature. An ML model could be used to learn and explain the dominance rankings in a dominance-based method, or to learn and explain different decomposition strategies in decomposition-based methods. In addition to supporting DMs, these explanations could serve the researchers or analyst operating and implementing the methods to build a better understanding of which parameters to choose to achieve the best possible performance. If explainability is not a goal, then exploring the potential of the more powerful ML models, such as deep neural networks and support vector machines, could also be explored further in LEMOO methods.

Our study was also limited to rule-based explanations. In future studies, exploring different kinds of explanations, such as causal and counterfactual explanations [Molnar 2022], would be interesting. Moreover, explaining aspects of a MOO problem other than the characteristics of decision vectors near a reference point should be studied further. For instance, explaining the connection between preferences and computed solutions, as was done in the work of Misitano et al. [2022] for reference point methods, could be pursued in the future studies of LEMOOs.

The future directions listed here, and especially in Section 6.3, are nothing but the tip of the iceberg. However, the XLEMOO framework discussed in Section 3.3 provides other researchers and practitioners alike with a solid starting point to explore the preceding ideas and more. Because our framework is openly available and implemented as open source software, everyone is free

to utilize and apply it, or extend it for others to use as well. This highlights the importance of providing similar, open source, and openly available frameworks in future studies as well, and collaborating on open source software to promote the openness and renewal of research.

7 CONCLUSION

We explored the potential of applying LEMs to solve MOO problems through our implementation of a LEMOO method. We explored the performance aspects of our LEMOO method and showcased the potential it can have in supplying a DM with additional explanations providing them insights about the characteristics of decision variables near a solution of interest. We also discussed and provided the openly available XLEMOO framework for others to utilize and implement their own LEMOO methods with explanations.

Naturally, our work came with limitations, as discussed in Section 6, but also showed great potential in terms of future research. LEMs are, without a doubt, an understudied area in the field of EMO. Furthermore, when an interpretable ML model is utilized in a LEM's learning mode, we have the potential access to more insights about the characteristics of the populations generated during the course of an EMO method. These insights can be expressed in the form of explanations, as we did. To help the rest of the EMO community pursue their own research in studying the application of LEMs in an EMO context, we provide the discussed XLEMOO framework as openly available open source software.

Combining ideas of LEMs with ideas from EMO into LEMOO methods has vast potential to unveil new and exciting ideas in the field of MOO. Moreover, utilizing interpretable ML models in a learning phase in a LEMOO method has the potential to unlock additional insights thanks to the vast number of individuals generated in a typical EMO setting. By leveraging these insights into explanations, we have taken the first step toward establishing a new paradigm in the field of MOO: explainable and learnable MOO. It is our hope that our work will inspire and enable future studies and research to help develop this new field further.

A APPENDIX

Here, we present the main results of the experiments conducted in Section 4.2 for the car-side impact problem. The ideal and nadir points of the problem were approximated from a payoff-table (Table 4). The reference point used for the car-side impact problem was $\bar{\mathbf{z}}_{\text{car-side}} = (20.0; 3.5; 11.0; 0.1)$. The plots showing the aggregated measured quantities as a function of the switching frequency, are shown in Figures 11 and 12. The heatmaps for the aggregated measured quantities are shown in Figures 13 through 16. Refer to Section 4.2 for further details.

Table 4. Computed Pay-off Table for the Car-Side Impact Problems

Car-side impact problem				
Minimized	f_1	f_2	f_3	f_4
f_1	**15.58**	<u>4.43</u>	<u>13.09</u>	2.82
f_2	36.71	**3.59**	11.87	14.19
f_3	<u>39.05</u>	4.05	**10.61**	<u>29.81</u>
f_4	23.72	4.30	12.91	**0.14**

The first column indicates which objective has been minimized on each row. The minimum values in each column for each problem are in bold, whereas the maximum values are underlined.

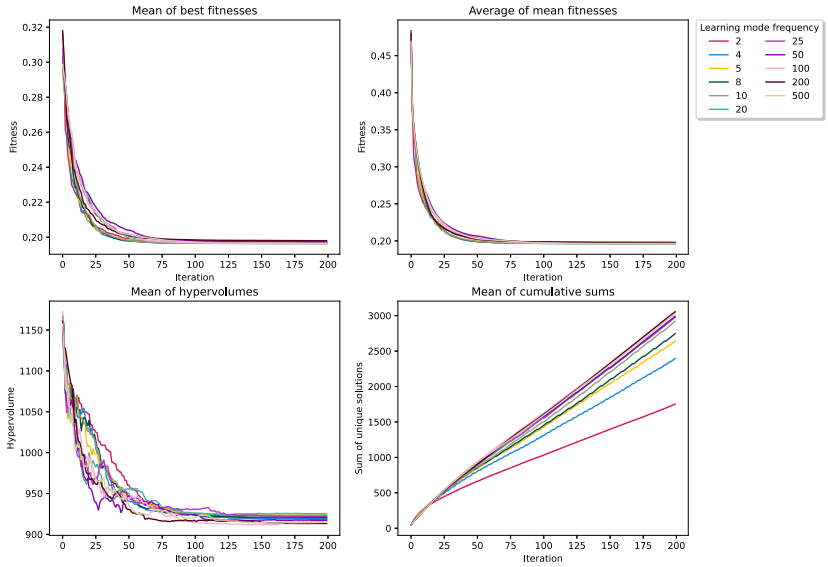

Fig. 11. The aggregated measured data for the vehicle crash worthiness problem after 200 iterations. The data is plotted for all the switching frequencies considered. The H/L split was kept constant at 20.

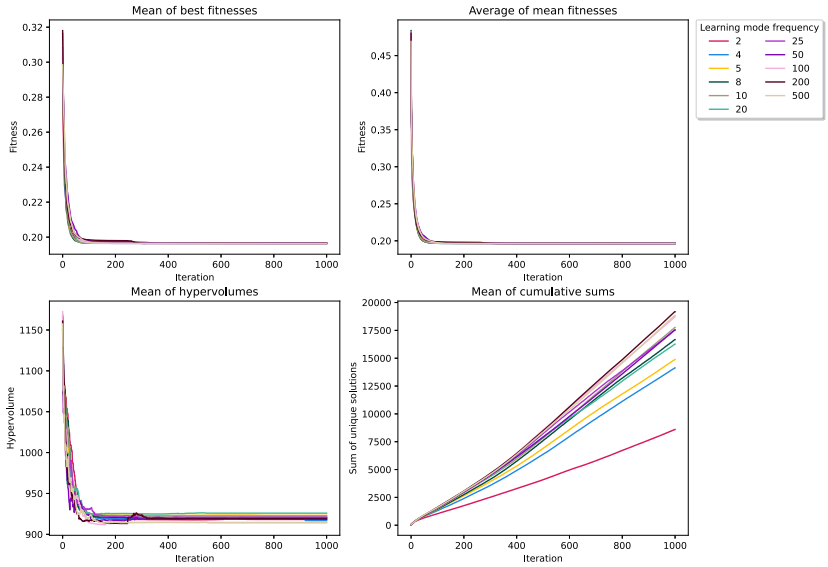

Fig. 12. The aggregated measured data for the car-side impact problem after 1,000 iterations. The data is plotted for all the switching frequencies considered. The H/L split was kept constant at 20.

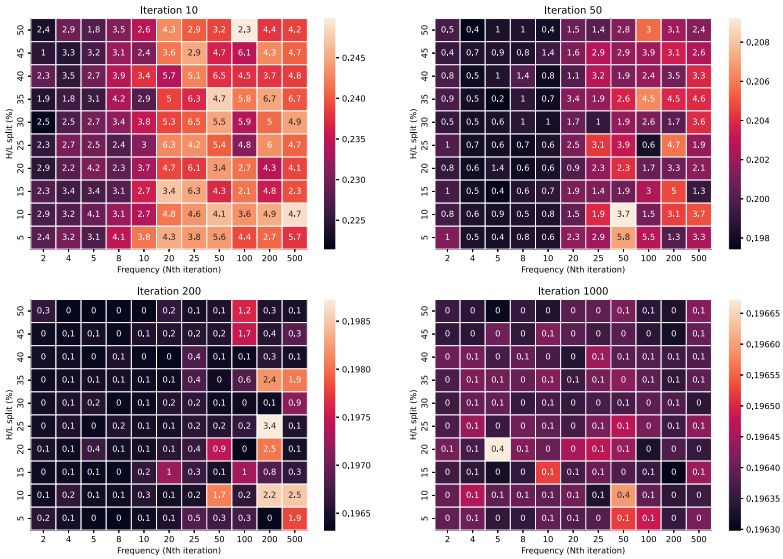

Fig. 13. The mean and standard deviations of the best fitness values for iterations 10, 50, 200, and 1,000 for the car-side impact problem.

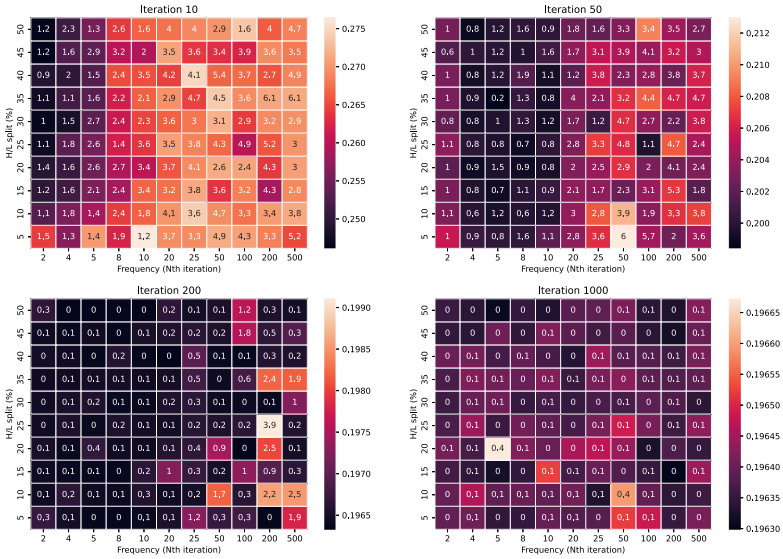

Fig. 14. The mean and standard deviations of the average of the fitness values for iterations 10, 50, 200, and 1,000 for the car-side impact problem.

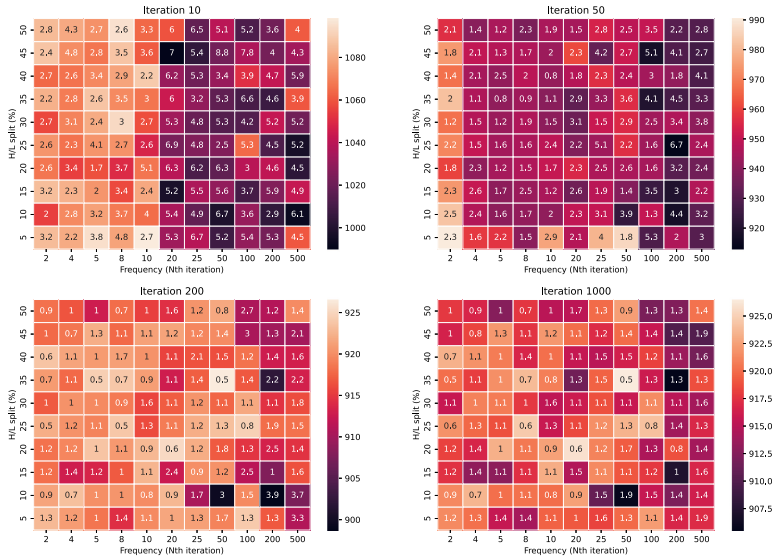

Fig. 15. The mean and standard deviations of the hypervolumes values of the population in iterations 10, 50, 200, and 1,000 for the car-side impact problem.

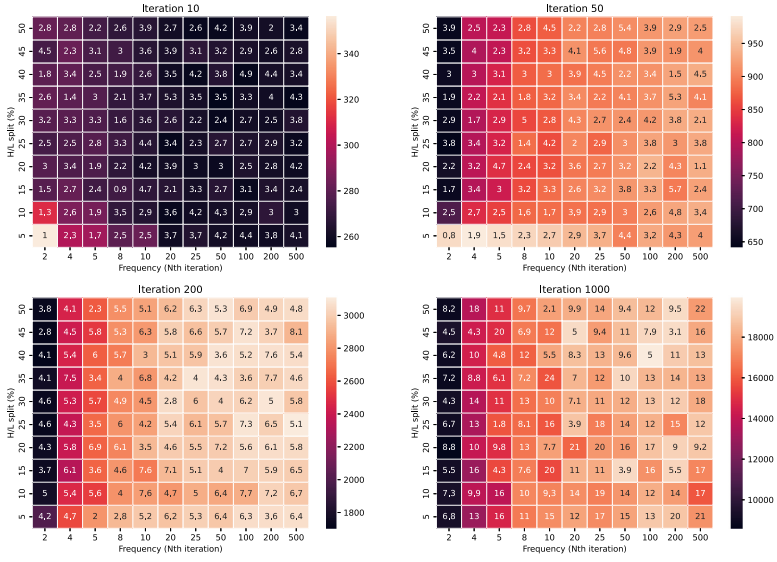

Fig. 16. The mean and standard deviations of the cumulative sums of unique solutions in iterations 10, 50, 200, and 1,000 for the car-side impact problem.

ACKNOWLEDGMENTS

I would like to warmly thank Kaisa Miettinen, Bekir Afsar, Francisco Ruiz, and Babooshka Shavazipour for their valuable comments on the manuscript. This research is related to the thematic research area Decision Analytics Utilizing Causal Models and Multiobjective Optimization (DEMO, jyu.fi/demo) at the University of Jyväskylä.

REFERENCES

Bekir Afsar, Kaisa Miettinen, and Francisco Ruiz. 2021. Assessing the performance of interactive multiobjective optimization methods: A survey. *ACM Computing Surveys* 54, 4 (2021), 1–27.

Bekir Afsar, Johanna Silvennoinen, and Kaisa Miettinen. 2023. A systematic way of structuring real-world multiobjective optimization problems. In *Evolutionary Multi-Criterion Optimization*. Lecture Notes in Computer Science, Vol. 13970. Springer, 593–605.

Bekir Afsar, Johanna Silvennoinen, Giovanni Misitano, Francisco Ruiz, Ana B. Ruiz, and Kaisa Miettinen. 2022. Designing empirical experiments to compare interactive multiobjective optimization methods. *Journal of the Operational Research Society* 2022 (2022), 1–12.

Luis Miguel Antonio and Carlos A. Coello Coello. 2017. Coevolutionary multiobjective evolutionary algorithms: Survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation* 22, 6 (2017), 851–865.

Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. 2020. Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion* 58 (2020), 82–115.

Sunith Bandaru, Amos H. C. Ng, and Kalyanmoy Deb. 2017a. Data mining methods for knowledge discovery in multiobjective optimization: Part A—Survey. *Expert Systems with Applications* 70 (2017), 139–159.

Sunith Bandaru, Amos H. C. Ng, and Kalyanmoy Deb. 2017b. Data mining methods for knowledge discovery in multiobjective optimization: Part B—New developments and applications. *Expert Systems with Applications* 70 (2017), 119–138.

Valerie Belton, Jürgen Branke, Petri Eskelinen, Salvatore Greco, Julián Molina, Francisco Ruiz, and Roman Słowiński. 2008. Interactive multiobjective optimization from a learning perspective. In *Multiobjective Optimization*. Springer, 405–433.

R. Benayoun, J. de Montgolfier, J. Tergny, and O. Laritchev. 1971. Linear programming with multiple objective functions: Step method (stem). *Mathematical Programming* 1, 1 (1971), 366–375.

Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer, New York, NY.

Guido Bologna. 2021. A rule extraction technique applied to ensembles of neural networks, random forests, and gradient-boosted trees. *Algorithms* 14, 12 (2021), 339.

Jurgen Branke, Jürgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Roman Slowiński. 2008. *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Vol. 5252. Springer Science & Business Media.

Leo Breiman. 2017. *Classification and Regression Trees*. Routledge.

Tinkle Chugh, Karthik Sindhya, Jussi Hakanen, and Kaisa Miettinen. 2019. A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms. *Soft Computing* 23, 9 (2019), 3137–3166.

Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. 2007. *Evolutionary Algorithms for Solving Multi-objective Problems*. Springer.

William W. Cohen and Yoram Singer. 1999. A simple, fast, and effective rule learner. In *Proceedings of the 16th National Conference on Artificial Intelligence and the 11th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI '99)*. 335–342.

Salvatore Corrente, Salvatore Greco, Benedetto Matarazzo, and Roman Slowinski. 2021. Explainable interactive evolutionary multiobjective optimization. *SSRN*. Retrieved October 23, 2023 from https://ssrn.com/abstract=3792994

Kalyanmoy Deb and Ram Bhushan Agrawal. 1995. Simulated binary crossover for continuous search space. *Complex Systems* 9, 2 (1995), 115–148.

Kalyanmoy Deb and Kaisa Miettinen. 2009. A review of nadir point estimation procedures using evolutionary approaches: A tale of dimensionality reduction. In *Proceedings of the Multiple Criterion Decision Making Conference (MCDM '08)*. 1–14.

Kalyanmoy Deb, Kaisa Miettinen, and Shamik Chaudhuri. 2010. Toward an estimation of nadir objective vector using a hybrid of evolutionary and local search approaches. *IEEE Transactions on Evolutionary Computation* 14, 6 (2010), 821–841.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and Tamt Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

Kalyanmoy Deb and Aravind Srinivasan. 2006. Innovization: Innovating design principles through optimization. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. 1629–1636.

Michael T. M. Emmerich and André H. Deutz. 2018. A tutorial on multiobjective optimization: Fundamentals and evolutionary methods. *Natural Computing* 17, 3 (2018), 585–609.

Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. 2017. Dermatologist-level classification of skin cancer with deep neural networks. *Nature* 542, 7639 (2017), 115–118.

Yoav Freund and Robert E. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* 55, 1 (1997), 119–139.

Nicolas Goix, Vighnesh Birodkar, Florian Gardin, Jean-Matthieu Schertzer, Hoebin Jeong, Manoj Kumar, Alexandre Gramfort, Tim Staley, Tom Dupré la Tour, Boyuan Deng, C, Fabian Pedregosa, Lawrence Wu, Ariel Rokem, Kyle Jackson, and mrahim. 2020. scikit-learn-contrib/skope-rules v1.0.1. Retrieved October 23, 2023 from https://doi.org/10.5281/zenodo.4316671

Bryce Goodman and Seth Flaxman. 2017. European union regulations on algorithmic decision-making and a "right to explanation." *AI Magazine* 38, 3 (2017), 50–57.

David Gunning and David Aha. 2019. DARPA's explainable artificial intelligence (XAI) program. *AI Magazine* 40, 2 (2019), 44–58.

Jussi Hakanen, Kaisa Miettinen, and Kristian Sahlstedt. 2011. Wastewater treatment: New insight provided by interactive multiobjective optimization. *Decision Support Systems* 51, 2 (2011), 328–337.

Jussi Hakanen, Kristian Sahlstedt, and Kaisa Miettinen. 2013. Wastewater treatment plant design and operation under multiple conflicting objective functions. *Environmental Modelling & Software* 46 (2013), 240–249.

Ching-Lai Hwang and Abu Syed Md. Masud. 1979. *Multiple Objective Decision Making: Methods and Applications*. Springer, Berlin, Germany.

Himanshu Jain and Kalyanmoy Deb. 2013. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: Handling constraints and extending to an adaptive approach. *IEEE Transactions on Evolutionary Computation* 18, 4 (2013), 602–622.

Laetitia Jourdan, David Corne, Dragan Savic, and Godfrey Walters. 2005. Preliminary investigation of the 'learnable evolution model' for faster/better multiobjective water systems design. In Evolutionary Multi-Criterion Optimization. Lecture Notes in Computer Science, Vol. 3410. Springer, 841–855.

Uday Kamath and John Liu. 2021. *Explainable Artificial Intelligence: An Introduction to Interpretable Machine Learning*. Springer.

Adhe Kania, Juha Sipilä, Giovanni Misitano, Kaisa Miettinen, and Jussi Lehtimäki. 2022. Integration of lot sizing and safety strategy placement using interactive multiobjective optimization. *Computers & Industrial Engineering* 173 (2022), 108731.

Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. 2016. Jupyter notebooks—A publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt (Eds.). IOS Press, 87–90.

Xingtao Liao, Qing Li, Xujing Yang, Weigang Zhang, and Wei Li. 2008. Multiobjective optimization for crash safety design of vehicles using stepwise regression model. *Structural and Multidisciplinary Optimization* 35, 6 (2008), 561–569.

Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. 2020. Explainable AI: A review of machine learning interpretability methods. *Entropy* 23, 1 (2020), 18.

Zachary C. Lipton. 2018. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue* 16, 3 (2018), 31–57.

Ruonan Liu, Boyuan Yang, Enrico Zio, and Xuefeng Chen. 2018. Artificial intelligence for fault diagnosis of rotating machinery: A review. *Mechanical Systems and Signal Processing* 108 (2018), 33–47.

Jie Lu, Guangquan Zhang, Da Ruan, and Fengjie Wu. 2007. *Multi-Objective Group Decision Making*. Imperial College Press.

Scott M. Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS '17)*. 4768–4777.

Michael D. McKay, Richard J. Beckman, and William J. Conover. 2000. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics* 42, 1 (2000), 55–61.

Ryszard S. Michalski. 2000. Learnable evolution model: Evolutionary processes guided by machine learning. *Machine Learning* 38, 1 (2000), 9–40.

Kaisa Miettinen. 1999. *Nonlinear Multiobjective Optimization*. Kluwer Academic, Boston, MA.

Kaisa Miettinen, Jussi Hakanen, and Dmitry Podkopaev. 2016. Interactive nonlinear multiobjective optimization methods. In *Multiple Criteria Decision Analysis*. Springer, 927–976.

Kaisa Miettinen and Marko M. Mäkelä. 1999. Comparative evaluation of some interactive reference point-based methods for multi-objective optimisation. *Journal of the Operational Research Society* 50, 9 (1999), 949–959.

Kaisa Miettinen and Marko M. Mäkelä. 2002. On scalarizing functions in multiobjective optimization. *OR Spectrum* 24, 2 (2002), 193–213.

Kaisa Miettinen and Francisco Ruiz. 2016. NAUTILUS framework: Towards trade-off-free interaction in multiobjective optimization. *Journal of Business Economics* 86 (2016), 5–21.

Kaisa Miettinen, Francisco Ruiz, and Andrzej P. Wierzbicki. 2008. Introduction to multiobjective optimization: Interactive approaches. In *Multiobjective Optimization*. Springer, 27–57.

Giovanni Misitano. 2020. Interactively learning the preferences of a decision maker in multi-objective optimization utilizing belief-rules. In *Proceedings of the 2020 IEEE Symposium Series on Computational Intelligence (SSCI '20)*. IEEE, Los Alamitos, CA, 133–140.

Giovanni Misitano. 2023a. gialmisi/XLEMOO: Article Code. Retrieved October 23, 2023 from https://doi.org/10.5281/zenodo.8318957

Giovanni Misitano. 2023b. XLEMOO Numerical Experiment Data. Retrieved October 23, 2023 from https://doi.org/10.5281/zenodo.8085638

Giovanni Misitano, Bekir Afsar, Giomara Lárraga, and Kaisa Miettinen. 2022. Towards explainable interactive multiobjective optimization: R-XIMO. *Autonomous Agents and Multi-Agent Systems* 36, 2 (2022), 1–43.

Giovanni Misitano, Bhupinder Singh Saini, Bekir Afsar, Babooshka Shavazipour, and Kaisa Miettinen. 2021. DESDEO: The modular and open source framework for interactive multiobjective optimization. *IEEE Access* 9 (2021), 148277–148295.

Felix Mölder, Kim Philipp Jablonski, Brice Letcher, Michael B. Hall, Christopher H. Tomkins-Tinch, Vanessa Sochat, Jan Forster, Soohyun Lee, Sven O. Twardziok, Alexander Kanitz, et al. 2021. Sustainable data analysis with Snakemake. *F1000Research* 10 (2021), 33.

Christoph Molnar. 2022. *Interpretable Machine Learning* (2nd ed.). Christoph Molnar. https://christophm.github.io/interpretable-ml-book

Behzad Moradi. 2018. Multi-objective mobile robot path planning problem through learnable evolution model. *Journal of Experimental & Theoretical Artificial Intelligence* 31, 2 (Nov. 2018), 325–348.

Behzad Moradi. 2019. The new optimization algorithm for the vehicle routing problem with time windows using multi-objective discrete learnable evolution model. *Soft Computing* 24, 9 (2019), 6741–6769.

Behzad Moradi and Abdolreza Mirzaei. 2016. A new automated design method based on machine learning for CMOS analog circuits. *International Journal of Electronics* 103, 11 (2016), 1868–1881.

Deepak Nagar, Palaniappan Ramu, and Kalyanmoy Deb. 2022. Visualization and analysis of Pareto-optimal fronts using interpretable self-organizing map (iSOM). *Swarm and Evolutionary Computation* 76 (2022), 101202.

Sara Narteni, Melissa Ferretti, Vanessa Orani, Ivan Vaccari, Enrico Cambiaso, and Maurizio Mongelli. 2021. From explainable to reliable artificial intelligence. In *Proceedings of the International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer, 255–273.

Yunyun Niu, Detian Kong, Rong Wen, Zhiguang Cao, and Jianhua Xiao. 2021. An improved learnable evolution model for solving multi-objective vehicle routing problem with stochastic demand. *Knowledge-Based Systems* 230 (2021), 107378.

Tapabrata Ray, Hemant Kumar Singh, Kamrul Hasan Rahi, Tobias Rodemann, and Markus Olhofer. 2022. Towards identification of solutions of interest for multi-objective problems considering both objective and variable space information. *Applied Soft Computing* 119 (2022), 108505.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1135–1144.

Steven L. Salzberg. 1994. *C4. 5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993*. Kluwer Academic.

Yoshikazu Sawaragi, Hirotaka Nakayama, and Tetsuzo Tanino. 1985. *Theory of Multiobjective Optimization*. Elsevier.

Chandan Singh, Keyan Nasseri, Yan Shuo Tan, Tiffany Tang, and Bin Yu. 2021. imodels: A Python Package for Fitting Interpretable Models. Retrieved October 23, 2023 from https://doi.org/10.21105/joss.03192

Henrik Smedberg and Sunith Bandaru. 2022. Interactive knowledge discovery and knowledge visualization for decision support in multi-objective optimization. *European Journal of Operational Research* 306, 3 (2022), 1311–1329.

Ralph Steuer. 1989. *Multiple Criteria Optimization: Theory, Computation, and Application*. Krieger Publishing Company.

Roykrong Sukkerd, Reid Simmons, and David Garlan. 2018. Toward explainable multi-objective probabilistic planning. In *Proceedings of the 2018 IEEE/ACM 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS '18)*. IEEE, Los Alamitos, CA, 19–25.

El-Ghazali Talbi, Matthieu Basseur, Antonio J. Nebro, and Enrique Alba. 2012. Multi-objective optimization using metaheuristics: Non-standard algorithms. *International Transactions in Operational Research* 19, 1-2 (2012), 283–305.

Ryoji Tanabe and Hisao Ishibuchi. 2020. An easy-to-use real-world multi-objective optimization problem suite. *Applied Soft Computing* 89 (2020), 106078.

Jasper van der Waa, Elisabeth Nieuwburg, Anita Cremers, and Mark Neerincx. 2021. Evaluating XAI: A comparison of rule-based and example-based explanations. *Artificial Intelligence* 291 (2021), 103404.

Jinkun Wang, Yezheng Liu, Jianshan Sun, Yuanchun Jiang, and Chunhua Sun. 2016. Diversified recommendation incorporating item content information based on MOEA/D. In *Proceedings of the 2016 49th Hawaii International Conference on System Sciences (HICSS '16)*. IEEE, Los Alamitos, CA, 688–696.

Andrzej P. Wierzbicki. 1980. The use of reference objectives in multiobjective optimization. In *Multiple Criteria Decision Making Theory and Application*. Springer, 468–486.

Andrzej P. Wierzbicki. 1982. A mathematical basis for satisficing decision making. *Mathematical Modelling* 3, 5 (1982), 391–405.

Bin Xin, Lu Chen, Jie Chen, Hisao Ishibuchi, Kaoru Hirota, and Bo Liu. 2018. Interactive multiobjective optimization: A review of the state-of-the-art. *IEEE Access* 6 (2018), 41256–41279.

Huixin Zhan and Yongcan Cao. 2019. Relationship explainable multi-objective optimization via vector value function based reinforcement learning. *arXiv preprint arXiv:1910.01919* (2019).

Qingfu Zhang and Hui Li. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation* 11, 6 (2007), 712–731.

Eckart Zitzler and Simon Künzli. 2004. Indicator-based selection in multiobjective search. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*. 832–842.

Received 15 November 2022; revised 16 September 2023; accepted 28 August 2023