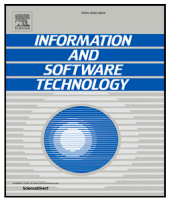
Information and Software Technology 152 (2022) 107031

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Variability testing of software product line: A preference-based dimensionality reduction approach

Thiago Ferreira [a,*], Silvia Regina Vergilio [b], Marouane Kessentini [c]

[a] College of Innovation & Technology, University of Michigan-Flint, Flint, USA
[b] Computer Science Department, Federal University of Paraná, Curitiba, Brazil
[c] School of Engineering and Computer Science, Oakland University, Rochester, USA

ARTICLE INFO

Keywords:
Spl testing
Search-based software engineering
Preference-based algorithms
Dimensionality reduction

ABSTRACT

Context: Multi- and many-evolutionary algorithms have been applied to derive products for the variability testing of Software Product Lines (SPLs). This problem refers to the selection of an adequate product set to test a SPL by optimizing some objectives related to the number of products to be tested, testing criteria to be satisfied, and revealed faults. However, some problems emerge when the number of objectives to be optimized increases, for example: the solutions generated by the optimization algorithms become incomparable, designing a Pareto-front in this context requires a large number of solutions, and the visualization of such solutions requires special techniques. Several techniques are proposed to tackle this problem, such as decomposition and algorithms based on indicators. Among them, algorithms based on dimensionality reduction and user preferences are widely used, but there are no studies in the literature investigating the usage of both in a combined way.

Objective: In light of this, we introduce COR-NSGA-II (Confidence-based Objective Reduction NSGA-II). COR-NSGA-II defines for each objective a confidence-level calculated with the user preferences provided interactively. The objectives with higher values of confidence are removed from the next algorithm execution.

Method: For assessing the feasibility of COR-NSGA-II, experiments were conducted by using six different SPLs, seven objectives, two types of reference points representing the user preferences, and two scenarios to simulate different user profiles.

Results: COR-NSGA-II is evaluated against four algorithms explored in the literature for the problem, and outperforms most of them according to R-HV and R-IGD. It takes less time to execute and generates a reduced number of solutions, all of them satisfying the user preferences.

Conclusion: A qualitative analysis performed with 12 potential users shows that the task of selecting a solution generated by COR-NSGA-II is easier than selecting a solution generated by the other algorithms.

1. Introduction

The Variability Testing of Software Product Line (VTSPL) refers to selecting test configurations – products – usually derived from the SPL variability model, such as the Feature Model (FM). Ideally, all possible products should be tested, but this is many times impracticable because the number of products grows exponentially with the number of features [1]. In this way, only a set of the most representative products should be selected, usually considering factors such as cost, the number of revealed faults, and tested features.

In the literature, the VTSPL problem has been addressed in the Search-based Software Engineering (SBSE) field [2], which applies search-based techniques to solve different optimization problems from the Software Engineering (SE) area. We can find many formulations for the VTSPL problem [3–12] where the authors suggest optimizing distinct objectives (we identified at least seven objectives) and applying different algorithms, mainly evolutionary ones. However, our recent paper [13] investigated seven objective functions for this problem, and the results showed that the optimization algorithms such as NSGA-III and PCA-NSGA-II took up to 15 h to find suitable solutions on large feature models such as Drupal [14]. Also, they generated many solutions in the final Pareto-front, which could lead the user to spend much time finding suitable solutions for his/her context. Consequently, the user could even reject such solutions since they were generated without his/her feedback.

Problems like this, impacted by three or more objectives are called many-objective and are very common in SE where most of them are

* Corresponding author.
E-mail addresses: thiagod@umich.edu (T. Ferreira), silvia@inf.ufpr.br (S.R. Vergilio), marouane@umich.edu (M. Kessentini).

https://doi.org/10.1016/j.infsof.2022.107031
Received 15 September 2021; Received in revised form 28 July 2022; Accepted 30 July 2022
Available online 6 August 2022

naturally complex and many conflicting objective functions need to be optimized at the same time, as pointed out by Mkaouer et al. [15]. However, a survey [16] reports that 50% of the proposed algorithms address SE problems from only a bi-objective perspective, 30% consider three objectives, and 20% of the existing studies address more than four objectives. Multi-Objective Evolutionary Algorithms (MOEAs) are the most applied [17].

As claimed by Mkaouer et al. [15], a possible reason why SE problems have not been formulated as many-objective is due to the challenges in building a many-objective solution. To aggregate some objectives into one objective is a simple approach that could be used. However, there is a loss of information in this approach due to, among several reasons, the conflicting nature of the quality metrics used to assess the algorithm's performance. Then, as this approach and traditional MOEAs are not sufficient, there is a growing demand for scalable SBSE approaches that address SE problems in which many objectives are considered. In this perspective, improving the scalability of SBSE approaches will increase their applicability in industry and real-world settings.

However, some problems arise when the number of objectives increases. Deb and Jain [18] state that selecting a solution gets harder because most solutions become incomparable. Many solutions are required to generate a Pareto-front, and this generation process takes much time and requires special techniques for visualizing the solutions. Thus, several techniques are proposed to address such problems, such as new preference ordering relations, decomposition (a general approach to solving a problem by breaking it up into smaller ones and solving each of the smaller ones separately, either in parallel or sequentially), and so on. Among them, one of the most-used techniques in the optimization field is dimensionality reduction, as described in [19].

In the context of dimensionality reduction, Many-Objective Evolutionary Algorithms (MaOEAs) are executed seeking to reduce the number of objectives by removing the redundant ones, that is, objectives where there may not exist any conflict among them. PCA-NSGA-II [20] is an example of MaOEAs, applied in SE problems, including to the VTSPL [13], which uses the concept of Principal Analysis Component jointly with the NSGA-II algorithm for reducing the number of objectives.

Li et al. [19] point out that dimensionality reduction approaches have three main advantages: (i) to reduce computational load; (ii) to help decision makers to better understand the many-objective problem by pointing out the non-conflicting objectives; and (iii) to allow the combination with other approaches. However, the authors also state that if the addressed problem has just conflicting objectives, this one may limit the application of the approach once these algorithms may fail in reducing the number of objectives to be optimized or return a solution set that does not cover the complete Pareto-front. From this perspective, the combination of two or more approaches for tackling many-objective problems would be very interesting. For example, to take into account the user preferences, since human knowledge and judgment can be used to guide the search to reach the best solutions. However, as far as we know, there are no studies in the literature combining both approaches in an interactive way (or in-the-loop). Thus, this work intends to explore possible advantages of incorporating the user preferences provided interactively during the search for the reduction of objectives in many-objective optimization.

To achieve the above-mentioned goal, this work introduces an algorithm called COR-NSGA-II (Confidence-based Objective Reduction NSGA-II), which reduces the problem dimensionality guided by the user preferences. The user preferences about the solutions are captured interactively (or in-the-loop) by using an ordinal scale composed of three items *Non-preferred*, *No Opinion*, and *Preferred*. Based on them, a confidence level for each objective is determined and used to decide which objectives should be removed from the next execution of the NSGA-II algorithm.

COR-NSGA-II is evaluated for the VTSPL problem using six SPLs and a formulation with seven objectives. COR-NSGA-II is compared to MOEAs and MaOEAs used for the problem in the literature: NSGA-II, NSGA-III, R-NSGA-II, and PCA-NSGA-II. The results show that COR-NSGA-II is capable of guiding the search process to the objectives preferred by the users, by generating a small set of suitable solutions that incorporate the user preferences, and taking less time to execute. A qualitative analysis performed with a set of 12 potential users shows that for them, the selection of a solution generated by COR-NSGA-II is easier than the selection of a solution generated by the other algorithms. In this way, the main contributions of this paper are as follows:

- Introduction of COR-NSGA-II, a preference-based dimensionality reduction algorithm. It has some advantages concerning dimensionality reduction algorithms and is also capable of performing well in the presence of conflicting objectives, reducing the number of solutions to be visualized by guiding the search according to user preferences captured interactively. Then this makes easier the choice of a solution for the user.
- Evaluation results from the application of COR-NSGA-II to the VTSPL problem show that it generates quality solutions regarding the user preference indicators and spends less time to execute in comparison with MOEAs and MaOEAs. This makes the use of many-objective formulations easier in practice and contributes to increase the scalability of SBSE approaches and their adoption for real and complex problems.
- Implementation of COR-NSGA-II on Nautilus Framework [21], a tool that allows practitioners developing and experimenting several multi- and many-objectives evolutionary algorithms guided (or not) by human participation in a few steps with a minimum required background in coding and search-based algorithms. This makes easy the application of COR-NSGA-II to similar SE problems and its use in future research.

The paper is organized as follows: Section 2 provides background on the algorithms and quality indicators. Section 3 reviews the VTSPL problem, related work and problem formulation adopted in our work. Section 4 introduces COR-NSGA-II. Section 5 describes evaluation setup. Section 6 presents and analyses the obtained results; Section 7 discusses main implication and future work; Section 8 highlights the threats to validity; and Section 9 concludes.

2. Background

2.1. Software product line testing

A Software Product Line (SPL) can be defined as a set of common products from a particular market segment or domain [22]. Such products share some features, which represent functionality or a system capability that is relevant and visible to the end user [10]. The features can be common to all products derived from the SPL, but they can also be variable being found in only some of them. Thus, the Feature Model (FM) diagram is used for managing such variability in most SPL methodologies. This diagram is represented as a hierarchical arrangement through a tree, and it is used for representing all the SPL commonalities and variabilities, as shown in Fig. 1, that contains the SPL for the domain of Mobile Phone.

A product is given by a combination of features. Fig. 2(a) shows an example of a valid product that can be derived from the FM of Fig. 1, and Fig. 2(b) an invalid one. The latter is invalid because Calls is missing while being mandatory. Thus it should be present in all derived products.

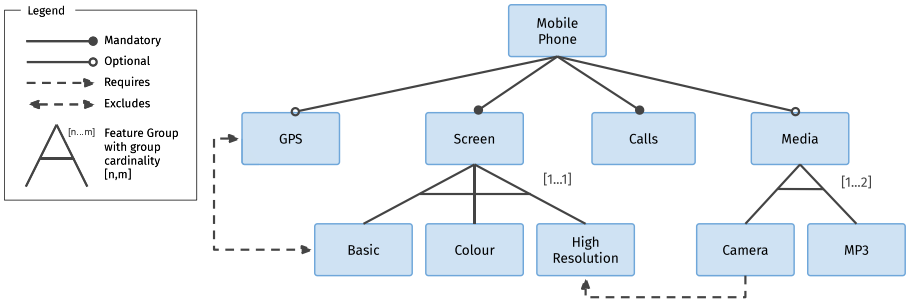


Fig. 1. Feature diagram of mobile phone.
Source: Adapted from [11].

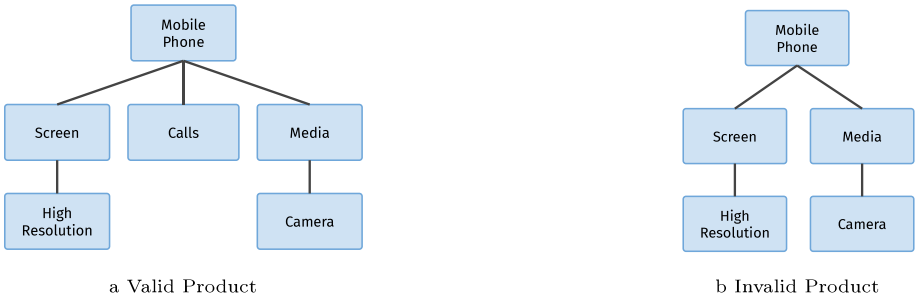


Fig. 2. Example of products generated from the FM in Fig. 1.

2.1.1. Pairwise testing in the FM context

In order to derive a set of products for the variability testing of SPLs, some studies in the literature are based on combinatorial testing [23–26]. Pairwise testing is one of the most popular kinds of combinatorial testing; therefore, it is also applied in our work. The goal of this testing criterion is to generate a set of products that includes all the valid pairs of features from the FM. Thus, the number of covered pairs can also be used for evaluating a set of products that were generated.

For instance, consider again the FM shown in Fig. 1. The pair (GPS, Basic) is invalid, and should not be required. Considering only the variabilities, we see that the product in Fig. 2(a) includes the pair (High Resolution, Camera) and does not include the pair (GPS, MP3). Thus, to derive the pairs, we use the Combinatorial tool[1] that implements the Automatic Efficient Test Generator (AETG) algorithm, introduced by Cohen et al. [27].

2.1.2. Mutation testing in the FM context

Another testing criterion that has been explored in the FM context [28–31] is mutation testing, a fault-based testing criterion. In the FM context, mutant FMs are generated with operators that describe possible faults that can be present in an FM. Hence, the goal of this testing criterion is to generate a product that is capable of distinguishing the behavior of the FM being tested from its mutant version.

Essentially, the product p is checked by an FM analyzer. The mutant is considered *dead* in two situations: (i) if p is *valid* according to the original FM and *invalid* for the mutant; and (ii) p is *invalid* for the original FM and valid for the mutant. When both FMs, original and mutant ones, validate the same set of products, they are considered as *equivalent*.

At the end of this process, a mutation score is calculated, given by the number of dead mutants over the total of non-equivalent generated mutants. Similar to the pairwise coverage described in the previous section, the score can be used for evaluating the adequacy of a set of products, or it can be used to improve an existing one. To illustrate this testing criterion, consider Fig. 3.

[1] http://161.67.140.42/CombTestWeb.

The figure shows that the operator changes a "requires" relation to an excludes one, such as the one between High Resolution and Camera. In this sense, the product in Fig. 2(a) kills the mutant, since it is *valid* for the original FM and it is *invalid* for the mutant.

2.2. Optimization algorithms

An optimization problem aims to find one or more feasible solutions which correspond to extreme values of one or more objectives (or objective functions) regarding the problem constraints [32]. The number of objective functions to be optimized defines which category the optimization problem belongs to. When an optimization problem involves a single-objective function, it is called Mono-objective Optimization Problem. When the number of objectives to be optimized holds two or three objective functions, the problem is called as Multi-objective Optimization Problem (MOP). Finally, when the number of objectives is four or more, the problem is called as Many-Objective Optimization Problem (MaOP).

Different types of algorithms to solve MOPs and MaOPs have been proposed in the literature [33]. Among them, it is possible to mention the Evolutionary Algorithms (EAs) [34]. These algorithms can find reasonably good approximations of the true Pareto-front in a reasonable time. A Pareto-front is a set of solutions where they are in such a way that the values of each objective cannot be improved without sacrificing the values of the other objective functions [35]. Some EAs are described as follows.

Multi-Objective Evolutionary Algorithms (MOEAs) are those based on Genetic Algorithm (GA) in which, by performing a stochastic optimization method, simulate the natural evolution process aiming to find solutions for MOPs. When the problem is a MaOP one, several Many-objective Evolutionary Algorithms (MaOEAs) are proposed in the literature to tackle such problems. Such algorithms include different categories, such as the algorithms based on dimensionality reduction and on user preferences. In this section, we describe the algorithms used in this work.

NSGA-II [36] creates in each interaction a new set of solutions (the offspring) based on their parents, joins the offspring with the

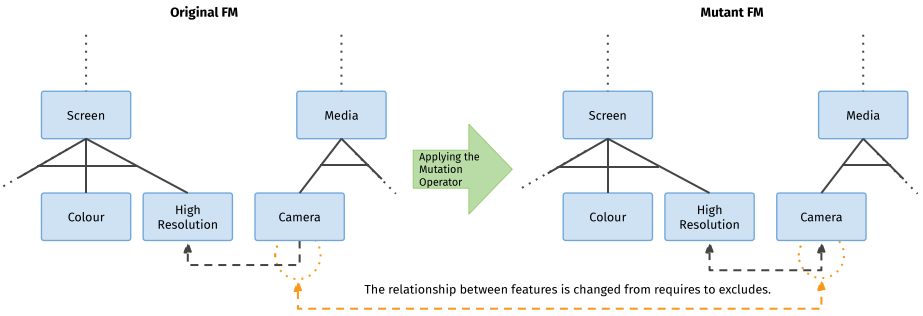


Fig. 3. Example of a mutant generated for FM in Fig. 1.

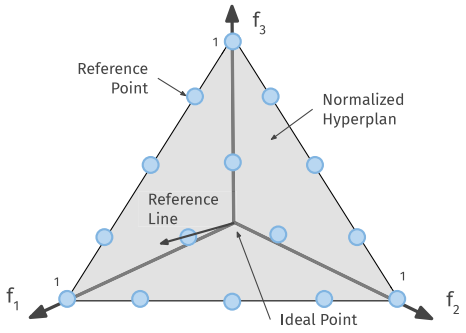


Fig. 4. Example of a normalized reference plane for a three-objective problem. *Source:* Adapted from [38].

parents, and sorts them according to their dominance rank and crowding distance. Therefore, in each generation, the solutions that are non-dominated when compared to the other ones survive to the next generation. A solution X dominates Y ($X \prec Y$) if X is at least as good as Y for all the objectives and strictly better than Y in at least one objective. Otherwise, X does not dominate Y [37]. When there are several non-dominated solutions and the next generation cannot receive all of them (since the number of desired solutions is defined a priori), the solutions with the greatest crowding distance (more scattered in the objective space) are selected to survive. Hence, this algorithm favors both the solutions that have the best fitness values (convergence) and the solutions that are more different (diversity). At the end of its execution, the algorithm returns the set of non-dominated solutions.

NSGA-III [18] basically replaces the crowding distance used by NSGA-II by a different one focused on a set of reference points Z^r (see Fig. 4). This mechanism helps to maintain the diversity among the solutions. In a generation t of NSGA-III, after the recombination, mutation, and non-dominated sorting, all acceptable fronts and the last front F_l that could not be completely included in P_{t+1} are included in a set RS_t. After that, the objective values and reference points are first normalized to be in an identical range. An orthogonal distance is computed between a member in RS_t and each of the reference lines (joining the ideal point and a reference point). After that, the solution is then associated with the reference point having the smallest orthogonal distance. The niche count ρ (defined as the number of solutions in S_t/F_l that are associated with the reference point) is computed for each reference point. Then, the reference point having the minimum niche count is identified, and the solutions from the last front F_l, associated with it, are included in the next population. At the final, the niche count of the identified reference point is increased by one, and the procedure is repeated to fill up the population P_{t+1}.

PCA-NSGA-II [20] applies an online dimensionality reduction using the PCA (Principal Component Analysis) method. This one is a multivariate technique that analyzes data in which observations are described by several inter-correlated quantitative dependent variables, where the goal is to extract the important information from the data and show the pattern of similarity of the observations and of the variables as points in maps [39]. Thus, PCA-NSGA-II identifies iteratively redundant objectives in the NSGA-II solutions as follows. Supposing we have M objective functions to be optimized and N population members, the initial data matrix X will be of size $M x N$. So, the method converts this matrix into a correlation matrix, in which negative numbers mean the objectives are negatively correlated (conflicting) and the positive ones mean the objective are positively correlated (redundant). After that, the method computes eigenvalues and eigenvectors and chooses non-redundant objectives by using a threshold cut (TC) value and the contribution for all principal components. Finally, if the current set of objectives has changed (it was reduced), the search of NSGA-II is performed and the PCA method is applied again. Otherwise, it returns the last Pareto-front found.

R-NSGA-II [40] requires one or more Reference Points (RPs), points in the search space where the user would like to concentrate the objectives. The RPs usually guide the search toward a Region of Interest (ROI), composed of non-dominated solutions preferred by the user. Moreover, the crowding distance metric is modified, being called "preferred distance". The use of this metric implies giving a greater emphasis to the solutions that are closer to the RPs provided. In order to maintain the diversity of selected solutions close to the RPs, R-NSGA-II applies a selection strategy called ϵ-clearing in which, by using a parameter named ϵ, the preferred distance is calculated as follows: (i) the normalized Euclidean distance of each boundary solution is calculated for each RP. The solutions are ranked in ascending order of distance. Thus, the solution with the smallest distance from RP is at the top of the rank; and (ii) after the calculation for all RPs, there will exist different rankings, one for each RP, and the preferred distance of a given solution will be the minimum one assigned to it considering all the rankings. Solutions with the smallest preferred distance values are selected and preferred to compose the new population.

3. Variability testing of software product line

The Variability Testing of Software Product Line (VTSPL) has as goal to ensure the products that can be derived from the variability model (commonly the Feature Model) satisfy their requirements. However, due to the complexity of the applications and the huge number of features and products available, only the most representative set of products is usually tested. The selection of the best products is an optimization problem addressed by many studies in the literature, mainly considering evolutionary algorithms. In this section, we describe these studies and the problem formulation adopted in our work.

3.1. Related work

Different formulations are tackling the variability testing problem in the literature by using combinatorial testing [41,42], as well as several papers addressing many objective functions to be optimized. Below we

describe existing works that focus on optimization algorithms (those more related to this work) grouped according to the kind of approach they apply.

Single-objective algorithms. These works usually deal with the selection of products by using an aggregation function where the optimized objectives are combined into a single function. The work of Wang et al. [3] propose a test case minimization approach. The authors use a GA and an aggregation function of the following factors: the number of test cases, pairwise coverage, and capability to reveal faults. In another work, Want et al. [4] propose a test case prioritization approach where it uses another aggregation function, including cost measures, and compares GA with (1+1) EA and random search. Ensan et al. [5] use a simple GA with an aggregation function comprised of cost and error rate factors. Henard et al. [6] also uses a GA with an aggregation function to handle the costs, pairwise coverage, and the number of products. All of them are conflicting objectives in the selection of test products. In another work of the authors, mutation testing is used to generate dissimilar products, that is, products that include different features [7].

Multi-objective Evolutionary Algorithms (MOEAs). Due to the characteristic of the VTSPL problem, MOEAs have presented better results than single-objective algorithms, where they usually adopt evolutionary algorithms. Lopez-Herrejon et al. [8] propose a multi-objective approach considering pairwise coverage and the size of the test suites. The approach of Matnei-Filho and Vergilio [9] considers objectives related to the mutation score, pairwise coverage, and number of products. The obtained results show no significant differences between the MOEAs used algorithms. However, NSGA-II obtained the best performance regarding the MOEA's quality attributes for the general case, compared to SPEA2 and IBEA. Strickler et al. [10], and Ferreira et al. [11] explore the use of hyper-heuristic approaches with multi-objective algorithms. The objectives used include pairwise, number of alive mutants, number of products, and similarity of products. The use of hyper-heuristics can help in the construction of more generic solutions and ease the implementation of the algorithms.

Many-objective Evolutionary Algorithms (MaOEAs). These algorithms deal better with the problem in the presence of three or more objectives. In this category, most works apply preference-based algorithms. Jakubovski Filho et al. [12] compared the NSGA-II algorithm with the preference-based algorithms r-NSGA-II and R-NSGA-II. Three- and four-objective formulations were used, including pairwise, number of alive mutants, number of products, and similarity of products. The results show that r-NSGA-II and R-NSGA-II outperformed NSGA-II. Moreover, the authors conclude that R-NSGA-II was the best option in most cases. More recently, Ferreira et al. [13] addressed the VTSPL problem by using NSGA-III and PCA-NSGA-II, with a set of seven objectives obtained from different approaches in the literature. The results show that NSGA-III reaches the best results regarding the quality indicators for all instances, but it takes longer to execute.

We observe that MOEAs and MaOEAs show the best results for the VTSPL problem. NSGA-II is the MOEA that presented the best general performance [9]. R-NSGA-II stands out among the preference-based algorithms [12]. NSGA-III reaches the best results regarding some quality indicators, but it takes a longer time compared to PCA-NSGA-II [13]. It is difficult to point out the best algorithm. For instance, R-NSGA-II does not provide an interactive way to provide the preferences and depends on the RP provided. NSGA-III takes longer to execute and produces a lot of solutions (requiring specific techniques to visualize and select solutions). PCA-NSGA-II does not have a good performance when all the objectives are not redundant. Using these algorithms in a combined way can be beneficial in overcoming some of those limitations, which is our work's goal. We have not found studies in the literature exploring the advantages of incorporating the user preferences provided interactively during the search for the reduction of objectives in many-objective optimization.

Moreover, a large number of objectives is used for the VTSPL problem. The most used are the number of products (test cases), pairwise coverage, mutation score (related to the capability to reveal faults), similarity of products regarding the features they contain or importance, and cost. In our work, we mapped all these objectives explored in the literature and derived a set of seven objectives, which are presented in the next section.

3.2. Problem formulation adopted

In this work, we adopt the formulation proposed by Jakubovski-Filho et al. [12], and the set of objective functions is derived considering different approaches from the literature described in the last section. Both the solution representation and the objective functions are described as follows.

A solution in the VTSPL problem is defined as a subset of products from all possible products that can be generated from a given FM. Since it is a subset, a binary encoding is employed to represent a selection where each gene represents a product. When the ith bit is equal to 1, the product p_i belongs to the solution. Otherwise, the ith bit is equal to 0.

We use the same convention to represent the features in a product, that is, 1 means the corresponding feature is selected for the product, otherwise, 0 means the feature is not selected. However, this representation is not part of the solution since several solutions have similar products selected (and consequently similar features).

Let $P = \{p_1, p_2, p_3, \ldots, p_n\}$ be a set of valid products being considered for the addressed FM, and $S \subseteq P$ be an solution, generated by an algorithm, with $|S|$ means the selected products. To generate P, we use the framework FaMa [43], which also gives the total number of products n. However, it is not possible to determine all the products for huge FMs. In this case, the tester provides a desired value for n, and n valid products are generated at random.

We adopted a set of seven objective functions, as presented in Table 1, which shows a description of the objectives and how they are calculated. All objective functions are normalized in $[0, 1]$, where 0 is the best value and 1 is the worst, that is, all of them should be minimized.

To clarify how the objective functions are computed, we consider an instance example with illustrative values for three features ($F1$, $F2$, and $F3$), in which all of them are non-mandatory ones. Their cost and importance are described in Table 2, and a set of five possible products that should be selected, shown in Table 3.

Now, consider that a solution $S = \{p_3, p_4\}$ was selected to be evaluated in which Products #3 and #4 were selected, the objective functions are calculated as follows:

$$
\begin{aligned}
N(S) &= \frac{2}{5} = 0.4 \\
M(S) &= 1.0 - \frac{4}{5} = 1.0 - 0.8 = 0.2 \\
P(S) &= 1.0 - \frac{3}{3} = 1.0 - 1.0 = 0.0 \\
V(S) &= \frac{2}{3} = 0.6 \\
C(S) &= \frac{22}{36} = 0.6 \\
F(S) &= 1.0 - \frac{3}{3} = 1.0 - 1.0 = 0.0 \\
I(S) &= 1.0 - \frac{11}{18} = 1.0 - 0.6 = 0.4
\end{aligned}
\tag{1}
$$

Therefore, in this instance example, the vector of objective values for the solution $S = \{p_3, p_4\}$ is (0.4, 0.2, 0.0, 0.6, 0.6, 0.0, 0.4).

Table 1
Objective functions.

Objective	Description	Rationale
Number of products $N(S) = \frac{\|S\|}{\|P\|}$	Ratio between the number of products in S and the number n of considered valid products in P	The # of products can impact the cost of the testing
Alive mutants $M(S) = 1.0 - \frac{KM}{AM}$	Where KM is the number of mutants killed by the products of S, and AM is the total number of alive mutants	A higher mutation score can help the bug identification
Uncovered pairs $P(S) = 1 - \frac{PC}{VP}$	Where PC is the number of pairs of features defining products covered by the products of S, and VP is the number of valid pairs	It evaluates the iteration among features
Products similarity $V(S) = \frac{RF}{OF}$	Where RF is the number of features that appears more than once in S and OF is the number of non-mandatory (optional) features in the instance. It takes into account the similarity between the products regarding the features they have	Similar products could let some features uncovered under testing
Products cost $C(S) = \frac{\sum_{p_i \in S} cost(p_i)}{\sum_{p_j \in P} cost(p_j)}$	Ratio of the cost of the selected products by the cost of all valid products, where $cost(p_i)$ returns the cost of a product p_i, estimated by summing the cost of its features	The development of expensive products impacts the cost of the testing. In this objective, the lower the value, the better.
Unselected features $F(S) = 1.0 - \frac{NF}{TF}$	Where NF is the number of features of S and TF represents the number of features the FM under test has. It measures how many features are in S	All features should be tested at least once
Unimportant features $I(S) = 1.0 - \frac{\sum_{p_i \in S} importance(p_i)}{\sum_{p_j \in P} importance(p_j)}$	Where $importance(p_i)$ returns the importance of the product p_i, given by the sum of the importance of the features in p_i. It returns the percentage of irrelevant features from the user's point of view	Allocating time and work to bring the most value to the company. In this objective, the greater the value, the better.

Table 2
Features from the instance example.

Features	Cost	Importance
F1	2	1
F2	4	2
F3	6	3

Table 3
Products from the instance example.

#	Features	Killed mutants	Covered pairs	Cost	Importance
p_1	[F1]	[M1]	[P2, P3]	2	1
p_2	[F1, F3]	[M1, M5]	[P2]	8	4
p_3	[F1, F2, F3]	[M3]	[P1, P2, P3]	12	6
p_4	[F2, F3]	[M1, M2, M4]	[P3]	10	5
p_5	[F2]	[M1, M3]	[P1]	4	2
Sum				**36**	**18**

4. Confidence-based objective reduction NSGA-II

COR-NSGA-II (Confidence-based Objective Reduction NSGA-II) is an algorithm that combines the advantages of preference and dimensionality reduction-based algorithms to deal with many-objective problems. It uses a confidence level for removing an objective from the next execution of the NSGA-II algorithm. The latter was chosen as the main search engine, especially for its good results in dealing with two or three objectives. A confidence level is defined based on the user preferences for each objective to be optimized, and these preferences are provided for values closest to the lowest and the highest values for each objective.

An overview of the algorithm is shown in Fig. 5. This shows that once a problem encoding (composed by a problem instance and a set of objectives to be optimized) is provided, COR-NSGA-II runs NSGA-II to find an approximate Pareto-front (composed by only non-dominated and non-duplicated solutions). Next, the solutions are shown to the user, and then s(he) has the opportunity to accept the found solutions or to provide his/her preferences aiming to reduce the number of objectives to be optimized. During the interaction process, the following components are taken into consideration:

- **Objective Values**: information for which the user is required to provide his/her preferences.
- ***Non-preferred, No Opinion, Preferred***: They are the preferences the user needs to provide about the objective values. If no preferences are provided, the default preference is *No Opinion*.
- **Confidence-based Selection**: the main method used for selecting or choosing the next subset of objectives to be optimized. This one takes into account the **Objective Values** and the preferences provided by the user.

To illustrate this, consider that the user is visualizing the approximate Pareto-front shown in Fig. 6. In this figure, the circles represent the possible locations where the user can provide his/her preferences. For instance, if the user clicks on circle #1, he/she will be providing user preferences for Solution #5 regarding Objective #1.

Firstly, COR-NSGA-II requires that solutions belonging to the Pareto-front must be normalized in [0.0:1.0] before showing them to the user, and it assumes, initially, all objectives should be selected for the next search process, but some of them should be removed (or not included). This initial assumption is important because if no user preferences are provided, the same objectives should be selected for the next search process, and no one should be removed. In Fig. 6, considering 0.0 as the lowest objective value and 1.0 the highest one, Solution #4 has the lowest value for Objective #1, Solution #5 has the lowest value for Objective #2, and Solution #3 the lowest value for Objective #3. On the contrary, Solution #5 has the highest value for Objective #1, Solution #3 has the highest one for Objective #2, and Solution #1 has the highest value for Objective #3.

COR-NSGA-II needs the user feedback about the objective values found by the search process (circles numbered from 1 to 15). However, the user does not need to provide all of them but only those most important ones from his/her point of view. For instance, the user can provide his/her preferences just for the circles *4*, *5*, *11*, and *12*, or, the user can provide his/her feedback just for the extreme values (objectives values equals to either 0.0 or 1.0) such as the circles *1*, *2*, *3*, *13*, *14*, and *15*. Once the user feedback is provided, COR-NSGA-II selects those closest to the lowest and highest values for each objective and defines a confidence level for removing the objective from the next subset (Section 4.1). The user feedback required by COR-NSGA-II must be composed of:

User Feedback = [solution index | objective index | objective value | preference]

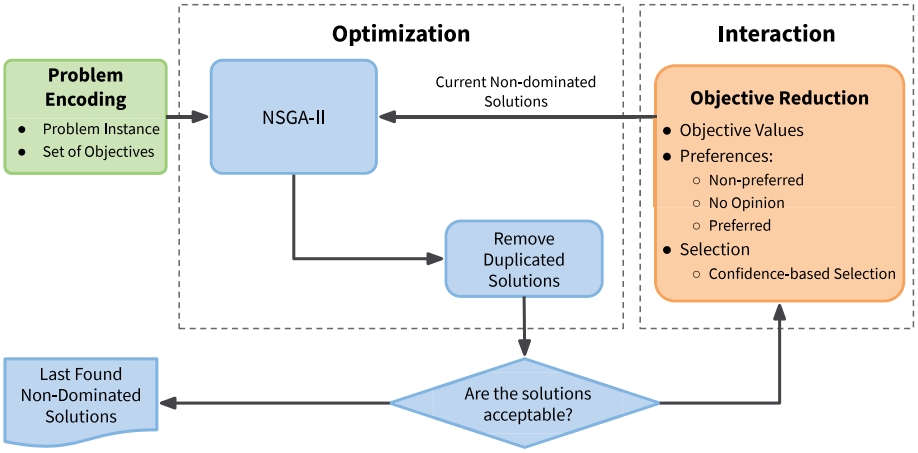


Fig. 5. COR-NSGA-II overview.

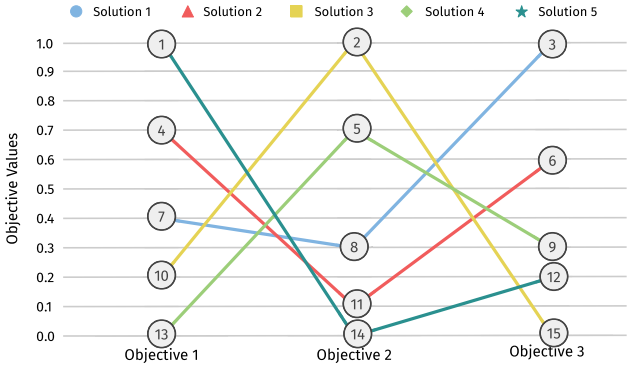


Fig. 6. Example of numbered objective values.

For instance, the user feedback [#1, #2, 0.0, *Preferred*] provided by the user means s(he) provided *Preferred* for Objective #2 with 0.0 from Solution #1, while [#2, #1, 0.7, *Non-preferred*] means s(he) provided *Non-preferred* for Objective #1 with 0.7 from Solution #2. Concluding, once all preferences are provided, the COR-NSGA-II's selection method takes action and uses the provided preferences to select the next subset of objectives to be optimized in the next execution. This method is described in more detail in the next section.

4.1. Confidence-based selection method

The Confidence-based Selection Method is responsible for defining which objectives should be removed from the next algorithm execution. The idea of this method is to calculate a removal confidence level for each objective and then, based on an input called $minConf$ (which represents the minimum removal confidence level), the method will decide if a given objective should be removed or not. That is, the objectives with the removal confidence levels greater or equal to $minConf$ are removed from the next algorithm execution. However, since the user preferences are provided as "*Non-preferred*", "*No Opinion*", and "*Preferred*", this method also translates all user preferences to an ordinal scale described in Table 4 before applying the $minConf$.

The correlation between the user preferences and a value is required because we have to define a priority among them. For instance, the user feedback [#1, #2, 0.0, *Preferred*], the user preference *Preferred* is converted to 1. So, in the case of more than one preference provided for the same objective value, we can choose the one with the lowest priority.

Table 4
Ordinal scale for the required information.

Required information	*Non-preferred*	*No Opinion*	*Preferred*
Value	−1	0	1

Algorithm 1 describes the procedure for selecting the next objectives. The algorithm takes as input the current population P, a set of optimized objectives, O, and a set of user feedback F.

Algorithm 1 Confidence-based Selection Method Algorithm

Input: The current population P
A set of optimized objectives $O = (o_1, o_2, \ldots, o_m)$
A set of user feedback $F = (f_1, f_2, \ldots, f_i)$
Output: A subset N of objectives to be optimized

Let $N \subset O$ be the next subset of objectives to be optimized
Let $MaxF = (maxF_1, maxF_2, \ldots, maxF_m)$ be the maximum feedback found by the algorithm for the optimized objectives, in which $\forall maxF_i \in MaxF, maxF_i \leftarrow NIL$
Let $MinF = (minF_1, minF_2, \ldots, minF_m)$ be the minimum feedback found by the algorithm for the optimized objectives, in which $\forall minF_i \in MinF, minF_i \leftarrow NIL$
Let $MinV = (minV_1, minV_2, \ldots, minV_m)$ be the minimum values for the population P.
Let $MaxV = (maxV_1, maxV_2, \ldots, maxV_m)$ be the maximum values for the population P.
for all f **in** F **do**
  i $\leftarrow$ $obj\_index(f)$
  distToMaxValue $\leftarrow$ $|maxV_i - obj\_value(f)|$
  distToMinValue $\leftarrow$ $|minV_i - obj\_value(f)|$
  if distToMaxValue < distToMinValue **then**
    if $maxF_i$ is NIL **or** $obj\_value(f) > obj\_value(maxF_i)$ **then**
      $maxF_i \leftarrow f$
    else if $obj\_value(f) \leftarrow obj\_value(maxF_i)$ **then**
      if $obj\_feedback(f) < obj\_feedback(maxF_i)$ **then**
        $maxF_i \leftarrow f$
      end if
    end if
  else if distToMinValue < distToMaxValue **then**
    if $minF_i$ is NIL **or** $obj\_value(f) < obj\_value(minF_i)$ **then**
      $minF_i \leftarrow f$
    else if $obj\_value(f) \leftarrow obj\_value(minF_i)$ **then**
      if $obj\_feedback(f) < obj\_feedback(minF_i)$ **then**
        $minF_i \leftarrow f$
      end if
    end if
  else if distToMinValue = distToMaxValue **then**
    if $minF_i$ is NIL **or** $obj\_feedback(f) < obj\_feedback(minF_i)$ **then**
      $MinF_i \leftarrow f$
    end if
    if $maxF_i$ is NIL **or** $obj\_feedback(f) < obj\_feedback(maxF_i)$ **then**
      $maxF_i \leftarrow f$
    end if
  end if
end for
$N \leftarrow$ the objectives selected by Algorithm 2, given O, $MaxF$, $MinF$
return N

Table 5
Confidence level for removing an objective.

		Highest value		
	Feedback	Preferred	No Opinion	Non-preferred
Lowest value	Preferred	0%	20%	0%
	No Opinion	80%	50%	20%
	Non-preferred	100%	80%	100%

In the first step (Lines 6–34), the algorithm goes through all user feedback trying to select the minimum and maximum feedback for each objective. For instance, it tries to find the feedback values for the most higher and lower objective values for each objective (again, all objectives are normalized in [0:1]). If two feedback values were provided for the same objective, the algorithm selects the one with a lower value.

As soon as the maximum and minimum are selected, the objective selection procedure is called (Algorithm 2).

Algorithm 2 Objective Selection Algorithm

Input: A minimum confidence level ($minConf$) value desired to remove an objective
A set of optimized objectives $O = (o_1, o_2, \ldots, o_m)$
A maximum feedback $MaxF = (maxF_1, maxF_2, \ldots, maxF_m)$ for all objectives
A minimum feedback $MinF = (minF_1, minF_2, \ldots, minF_m)$ for all objectives
Output: A subset N of objectives to be optimized
1: Let $N \subset O$ be the next subset of objectives to be optimized
2: Let $Lowest = (lowest_1, lowest_2, \ldots, lowest_m)$ be the confidence level for the solutions in the lowest objective values in which $\forall lowest_i \in Lowest, lowest_i \leftarrow 0$
3: Let $Highest = (highest_1, highest_2, \ldots, highest_m)$ be the confidence level for the solutions in the highest objective values in which $\forall highest_i \in Highest, highest_i \leftarrow 0$
4: **for** o_i **to** O **do**
5: **if** $maxF_i$ is not NIL **then**
6: $highest_i \leftarrow obj\_feedback(maxF_i)$
7: **end if**
8: **if** $minF_i$ is not NIL **then**
9: $lowest_i \leftarrow obj\_feedback(minF_i)$
10: **end if**
11: **if** confidence($lowest_i$, $highest_i$) $< minConf$ **then**
12: $N \leftarrow N \cup o_i$
13: **end if**
14: **end for**
15: **if** N is $\emptyset$ **then**
16: **return** a random objective from O
17: **end if**
18: **return** N

In this procedure, for each objective, the preferences provided by the minimum and maximum feedback are defined, respectively, to the lowest value and the highest one (Lines 4–9). After this step, the confidence level for removing this objective is calculated in Line 14 based on the information described in Table 5.

The values in Table 5 are defined based on the fact that the optimization algorithms try to minimize all objectives, and they are normalized in [0:1]. In this table, the lowest value means 0 and the highest one means 1. So, aiming to explain the table, consider that the user provided *Non-preferred* for both the lowest and highest values for a given objective. So, we assume with 100% of confidence level that this objective must be removed because the solutions generated with this objective tend not to be considered as good from the user's point of view. In the contrary, if both lowest and highest values are *Preferred*, we have to keep the objectives once the algorithm with them may still generate new good solutions. In another example, if the lowest value is *Non-preferred* and the highest is *Preferred*, we assume the algorithm tends to keep generating non-preferred solutions so we have to remove the corresponding objective with a 100% of confidence level.

Finally, we have to remove just the objectives in which the confidence level is greater than or equal to the $minConf$ that was previously defined by the user. However, if this method is carried out and all objectives should be removed (for example, when the user defines that the $minConf$ is 0%), a random objective must be picked up for the next algorithm execution.

To illustrate the method proposed, let us consider the example shown in Fig. 7 where the addressed problem has three objectives to be optimized, and the non-repeated and non-dominated solutions are composed of five solutions.

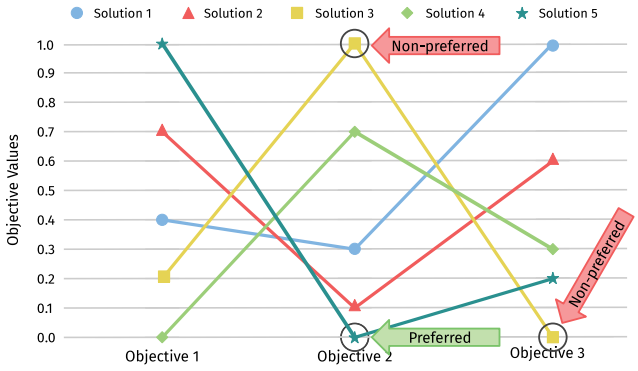


Fig. 7. Example of application of the confidence level.

Table 6
Confidence level example.

Objectives	Objective 1	Objective 2	Objective 3
Highest value	No Opinion	Non-preferred	No Opinion
Lowest value	No Opinion	Preferred	Non-preferred
Confidence level	50%	0%	80%

In this example, the user provided the following feedback. *Preferred* for Solution #5 in Objective 2; *Non-preferred* for Solution #3 in Objective 3; and *Non-preferred* for Solution #3 in Objective 3. The selection method aforementioned is performed by aiming to find the maximum feedback provided by each objective. In this example, there is no more than one feedback for the lowest and highest objective values. Then, the $Lowest$ and $Highest$ sets, and the confidence level (defined by the values in Table 5) are show in Table 6.

Supposing that $minConf$ defined by the user for removing an objective for the next execution is at least 80%, Objective 3 must be removed once it is associated with a confidence level of 80%. However, if $minConf$ is defined as 50%, Objectives 1 and 3 must be removed from the next execution.

This example shows another important property of this selection method. If $minConf$ is set to 100% (i.e., a very strict level), the method will only remove the objectives where the user provides either *Non-preferred* and *Preferred* for, respectively, the lowest and highest objective values, or *Non-preferred* for both. In an opposite way, if $minConf$ is 50% or less, if no user preferences are provided, the selection method considers that this objective is not good (once the user does not express his/her preferences about it), and it must be removed.

Finally, the confidence level described in this section is calculated by the algorithm, not provided by the user since the user preferences can be inconsistent after repeated sampling. Then, the only interaction that the users do is to select a solution and to provide if this solution is *Preferred*, *Non-preferred*, or *No Opinion*.

5. Empirical evaluation setup

COR-NSGA-II was implemented on Nautilus Framework [21] and applied to the VTSPL problem. Moreover, COR-NSGA-II was compared with MOEAs and MaOEAs used in the literature (see Section 2). To this end, the following Research Questions (RQs) were derived:

RQ1. Is COR-NSGA-II capable of reducing the problem dimensionality towards the user preferences? The goal of this RQ is to evaluate if the Confidence-based selection method of COR-NSGA-II is better than a random selection method. We applied the latter merely as a "sanity check" because all optimization algorithms should be capable of comfortably outperforming

Table 7
Characteristics of the FMs used in the experiments.

FM	# of products (n_t)	# of selected products (n)	Alive mutants (AM)	Valid Pairs (VP)	# of features (nf)
James [47]	68	68	106	75	14
CAS [48]	450	450	227	183	21
WS [49]	504	504	357	195	22
E-Shop [14]	1152	1152	94	202	22
Drupal [14]	≈2.09E9	11k	2194	1081	48
Smarthome [6]	≈3.87E9	11k	2948	1710	60

the random search for a well-formulated optimization problem. In this RQ, we executed both algorithms and the user preferences were provided by a simulated user (explained in more detail in Section 5.2). The results were compared using Reduction Efficiency, the Number of Preferred Objectives in the Last Subset, and Reduction Capacity (see Section 5.3).

RQ2. How are the results of COR-NSGA-II compared to those obtained by multi- and many-objective evolutionary algorithms? This RQ aims to compare the proposed algorithm to those that use reference-set-based, preference-based, or dimensionality reduction approaches for solving the VTSPL problem: NSGA-II, NSGA-III, R-NSGA-II, and PCA-NSGA-II. To reach this goal, the algorithm was executed using preferences provided by a simulated user. A quantitative analysis was performed by using R-HV, R-IGD, # of solutions generated (and the number of solutions in the ROI), and the execution time.

RQ3. Can COR-NSGA-II help users to find useful solutions? The goal of this research question is to evaluate if the solutions generated by COR-NSGA-II are more preferred than those generated by the other multi- and many-objective evolutionary algorithms. To this end, a set of potential users were invited and asked to select a good solution from their point of view. The analysis conducted is based on a qualitative questionnaire available in our supplementary material [44].

5.1. Target feature models

This work uses six FMs widely used in related work [6,10–12,45,46]. Table 7 shows information about the FMs: number of products (n_t), number of used products n, alive mutants (AM), valid pairs (VP), and number of features (nf). We can observe that the last two FMs contain a larger number of features and products. Due to this, it is impractical to use all the products in the population representation. For both FMs n products were randomly selected from the total number of products n_t that can be derived. Details about each FM can be found in our supplementary material [44].

In this work, James, CAS, WS, and E-Shop are the smallest instances, while Drupal and Smarthome are the largest ones. All FMs were used for evaluating **RQ1** and **RQ2**. E-Shop was chosen for answering **RQ3**, because it is composed of a reasonable number of products to be selected, and it has a suitable execution time for experiments where the users need to express their preferences. Also, in our work, we used the mutation operators of FMTS tool [30][2] which is responsible for generating the mutants and giving a set of products, calculating the mutation score and the number of alive mutants (AM).

5.2. Users

To evaluate our RQs, we used real and simulated users.

[2] FMTS works with the framework Feature Model Analyzer (FaMa) [43], which is responsible for handling constraints, and validating the mutants and products. This tool supports the FODA notation [50], extended and cardinality-based FMs.

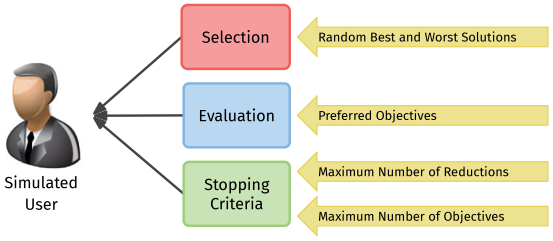


Fig. 8. Simulated user representation.

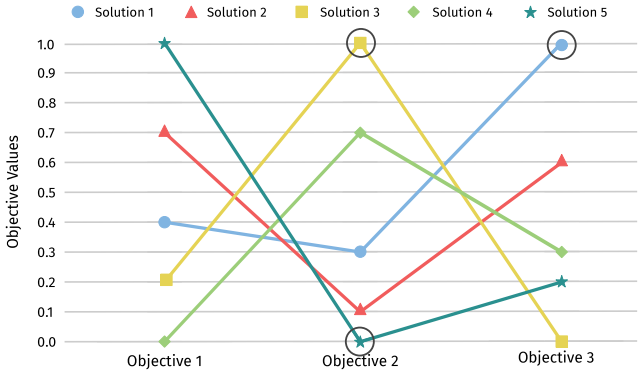


Fig. 9. Example for user simulator evaluation.

Simulated users. A user simulator was developed for answering **RQ1** and **RQ2**, aiming to represent a possible evaluation profile, as explored in other works in the literature [51–53]. In this method, the user simulator provides the preference for a given solution when required by the algorithm. It is important to note that the main objective of this simulator is not a faithful representation of a human being, but it demonstrates the influence of a certain evaluation profile in the search process. The user simulator requires a set of preferred objectives (a subset of those to be optimized) and it supposes that the population is normalized in [0:1] in which 0.0 means the best value and 1.0 is the worse one for every objective. So, this one is grouped into three main components: selection, evaluation, and stopping criteria briefly summarized in Fig. 8.

The first component is responsible for selecting the items for evaluation required by COR-NSGA-II. When this one is performed, for each objective, all solutions from the non-dominated population that have the best and the worst values are selected. After that, a set of random solutions from this group (in this context, called items for evaluation) is picked up to be evaluated later on by the user simulator.

In the second component, these items are evaluated. This one is responsible for evaluating the items proposed according to the preferred objectives previously defined for the user simulator. The algorithm first verifies if the objective index of the item for evaluation is part of the preferred objectives. If it is not, this item is marked as *Non-preferred*. Otherwise, it is marked as *Preferred* if the objective value is 0.0 or the maximum and minimum objective values are the same, and as *Non-Preferred* if the objective value is 1.0. In the last case, if no previous conditions were reached, it is marked as *No Opinion*.

Finally, the third component is responsible for defining the stopping criteria considered by the simulator. In this one, the used criteria are the maximum number of interactions or the maximum number of objectives is reached. To illustrate the user simulator, consider the non-dominated population shown in Fig. 9.

In this example, the preferred objectives are #1 and #2. So the items for evaluation are these ones:

$Item_1$ = [solution: #3 | objective: #2 | 1.0]

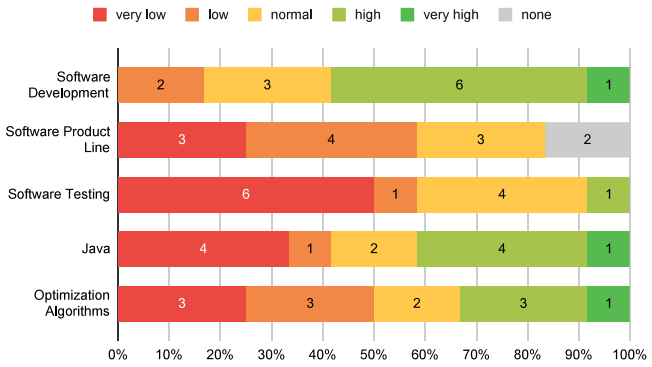


Fig. 10. Participants' experience.

$Item_2$ = [solution: #1 | objective: #3 | 1.0]

$Item_3$ = [solution: #5 | objective: #2 | 0.0]

Performing the evaluation component in this example, the user preferences provided by the user simulator for $Item_1$, $Item_2$, and $Item_3$ are respectively, *Non-preferred*, *Non-preferred*, and *Preferred*.

The user simulator requires a set of preferred objectives. Based on this, two scenarios were designed and evaluated in **RQ1** and **RQ2**. The first scenario (called *Scenario 2D*) is responsible for simulating a user who prefers two objectives, and the second one (called *Scenario 3D*) simulates a user who prefers three objectives. For *Scenario 2D*, *Number of Products* and *Alive Mutants* objectives are randomly defined as preferred ones. Regarding *Scenario 3D*, *Alive Mutants*, *Similarity*, and *Cost* objectives are selected as preferred ones.

The choice of these scenarios and the objectives selected for them were based on the correlation among the objectives, seeking to simulate a possible preferred set of objectives. For example, in *Scenario 2D*, all objectives are conflicting. Concerning to *Scenario 3D*, two of the selected objectives are redundant ones: *Similarity*, and *Cost*.

Real users. For answering **RQ3**, we asked a group of potential users to evaluate the solutions generated by COR-NSGA-II and other algorithms. They were invited to assist in the solution generation process by stating their preferences about the found solutions.

Our study involved 12 participants, including 3 Master students and 7 Ph.D. students in Software Engineering and Optimization Algorithms, and 2 professors. The participants were asked to fill out a questionnaire where helped to collect background information, such as their role within the company, their programming experience, and their familiarity with software testing.

The experiment involved 7 male and 5 female participants, and around 50% of them with high experience in software development and optimization algorithms (Fig. 10). In addition, all participants attended one lecture about the VTSPL problem and optimization algorithms. At the end, a test with 5 questions was filled out aiming to evaluate their performance and understanding of the subject for suggesting good solutions for an instance of an example problem.

Four groups with 3 participants were formed based on the pre-study questionnaire and test results in order to ensure the groups had participants with almost the same average skills. For the test results, each question was weighted, where we tried to form groups in which all users had a similar average. At the end, all participants reached a good performance by selecting the correct answer to almost all questions. Consequently, they were assigned randomly to each group since all of them were at the same level.

We asked every participant to define a set of preferred objectives based on his/her preferences. Then, the algorithms were executed by following the sequence described in each group. For each algorithm, the user was required to select/provide a good solution based on his/her preferences previously defined. In this scenario, a questionnaire was provided and the participants were asked to justify their evaluation about his/her decisions and these justifications are reviewed by the organizers of the study. Table 8 summarizes the group's organization and the order in which the algorithms were executed in the groups. For instance, the participants of Group 2 are invited to execute first NSGA-II, after R-NSGA-II and COR-NSGA-II, and at the end, to generate a solution for the problem instance by using manual selection. It is important to notice that the algorithms were executed with the same tooling. All questionnaires are available in [44].

Table 8
Groups organization.

Group 1	Group 2	Group 3	Group 4
Manual selection	NSGA-II	R-NSGA-II	COR-NSGA-II
NSGA-II	R-NSGA-II	COR-NSGA-II	Manual selection
R-NSGA-II	COR-NSGA-II	Manual selection	NSGA-II
COR-NSGA-II	Manual selection	NSGA-II	R-NSGA-II

5.3. Quality indicators

To calculate the quality indicators, we used three sets of solutions following optimization literature [54]: (i) PF_{approx}: set of non-dominated solutions obtained by one algorithm execution; (ii) PF_{known}: set of non-dominated solutions of an algorithm obtained by the union of all the PF_{approx}, removing the dominated and repeated ones; and (iii) PF_{true}: formed by all sets PF_{known} obtained from different algorithms by removing dominated and repeated solutions.

However, some regular indicators, such as Hypervolume (HV) and Inverted Generational Distance (IGD), do not take into account the RP informed by the user. Considering the great importance of this information when applying a preference-based algorithm, we decided to use the Hypervolume indicator with R-Metric (R-HV) and IGD with R-Metric (R-IGD) following Li et al. [19].

The general idea of R-Metric is to pre-process the preferred solutions according to a multi-criterion decision-making approach before using a standard metric to evaluate the performance of the obtained solutions. Fig. 11 presents an illustration of the steps applied by R-Metric calculation principle.

The first step is to filter the solutions by keeping only the non-dominated and no-repeated ones (Prescreening). In the second step (Pivot Point Identification), a representative point is identified, reflecting the general satisfaction of the solutions with respect to the RP. In the third step (Trimming), only solutions located in the ROI are of interest to the user. The R-Metric defines the ROI as a set of solutions that is centered at the pivot point and with length δ. Only solutions located in this approximated ROI are valid for performance assessment. After this, in the fourth step (Solution Transfer), the trimmed points are transferred to a virtual position to be evaluated its proximity to the RP. Finally, the last step (R-Metric Calculation) applies the quality indicator in the solutions processed by R-Metric. In our case, the HV and IGD quality indicators [54].

Fig. 12 shows an example of the application of the R-Metric for five Pareto-fronts, in which Fig. 12(a) shows the original Pareto-fronts and Fig. 12(b) the virtual ones after the application of the R-Metric.

It is possible to see that, for example, for the Pareto-front $S3$, the solutions in this one are closest to the RP, then the solutions remain almost in the same position in the search space. However, for Pareto-front $S5$, its virtual position is more distant from RP provided by the user. This will impact the quality attributes for this Pareto-front once the values are calculated by considering the virtual position. The objective of R-Metric is to evaluate the dissemination of solutions in the ROI and, at the same time, the proximity of these solutions to the RP. To answer **RQ1** and **RQ2** we use R-HV and R-IGD, and other indicators described as follows.

Fig. 11. R-Metric steps. *Source:* Adapted from [55].

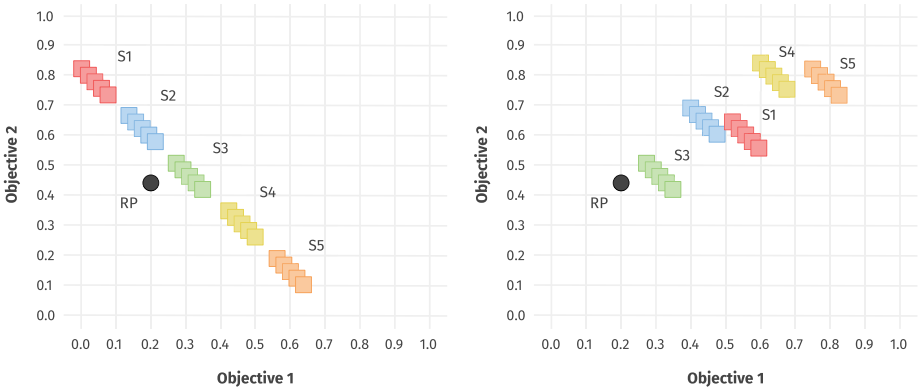


Fig. 12. R-Metric example.

- **Hypervolume with R-Metric (R-HV)**: is calculated considering the sets PF_{approx} generated by all algorithms [54]. At the end, the average of the results obtained by calculating the R-HV in each set is returned. Thus, higher values of R-HV present the best results, that is, results that contain a set of solutions closer to the RP and also contain a greater number of solutions with good diversity within the ROI.
- **Inverted Generational Distance with R-Metric (R-IGD)**: IGD is a convergence measure that corresponds to the average Euclidean distance between the Pareto-front approximation provided by an algorithm and a reference Pareto-front [19]. IGD needs an approximated (or real) Pareto-front to be calculated. Thus, some steps of R-Metric are performed on PF_{approx} such as the steps *pivot identification* and *trimming procedure*. The remaining solutions are considered as trimmed PF_{approx} and they are used to calculate the R-IGD quality attribute. So, the lower R-IGD, the better the results, that is, results that contain a set of solutions that are closer to the trimmed PF_{approx}.
- **Average Number of Solutions in the ROI**: calculates how well an algorithm can generate solutions in the ROI, that is, the main idea of this quality attribute is to evaluate the algorithm's ability to obtain concentrated solutions that satisfy the user preferences.
- **# of Targets in the Last Subset**: counts the number of preferred objectives in the last reduction. Ideally, this metric should return the same number of preferred objectives defined by the user to make sure the dimensionality reduction is moving towards user preferences.
- **Reduction Capacity**: calculates the percentage in which the reduction mechanism reached the preferred objectives. For instance, if this metric indicates that it has 100%, it means, in all executions, the algorithm reaches the preferred objectives defined by the user. Otherwise, this metric scores 0%. So, greater value are better.
- **Reduction Efficiency**: returns the number of reductions performed until the final set of objectives includes just the preferred ones. A lower number is better since the algorithm has a faster convergence, but 0 means the algorithm never reached the preferred objectives.
- **Execution Time**: measures the time spent in milliseconds on generating the final Pareto-front set. However, for preference-based algorithms such as COR-NSGA-II, the time spent on providing the user preferences is also taken into consideration. Thus, a lower number is better.

It is important to notice that COR-NSGA-II generates a non-dominated population with the same number of objectives used in the last reduction. So, it makes necessary to evaluate PF_{true} again with the same objectives used at the beginning of the execution. To cope with this, when COR-NSGA-II stops the reduction process, all solutions in PF_{true} are re-evaluated with the same objectives described in Section 3.2 and, so, it is possible to compare all algorithms once all of them have solutions with the same set of objectives.

5.4. Definition of the Reference Points (RP)

A reference point is a tuple RP = *(N, M, P, V, C, F, I)* where the sequence of elements represents either a value (or a point) in the objective space in which the ROI should be generated, or a point in which the algorithm should concentrate its search. So, in this tuple, N is the value for the objective *Number of Products*, M is the value for *Alive Mutants*, P is the value for *Uncovered Pairs*, V is the value for *Similarity*, C is the value for *Cost*, F is the value for *Unselected Features*, and I is the value for *Unimportant Features*.

Knowing this, two kinds of RPs were defined to be used by R-Metric and R-NSGA-II algorithm in **RQ2**: (i) restricted, and (ii) compromised. The former is responsible for representing a preference restricted to a specific set of preferred objectives, and to the other non-preferred objective "no preferences" are assigned. The latter is responsible for also representing a preference for a specific set of preferred objectives, but intermediate values for the other objectives are also defined. Thus, two RPs were defined for each scenario described for Simulated Users taking into account their preferred objectives.

Table 9 presents the RPs used to calculate the R-Metric and to run R-NSGA-II. For Scenario 2D, the restricted RP aims to express preferences for *Number of Products* and *Alive Mutants* with 0.0 value, and no preferences (a value of 1.0) for the other objectives, while in the compromised RP, 0.5 is used when there are no preferences. Concerning Scenario 3D, the RP aims to express the preference for *Alive Mutants*, *Similarity*, and *Cost* objectives with 0.0 value for the restricted RP and 1.0 for the non-preferred ones, while in the compromised RP, 0.5 is defined for the non-preferred objectives.

Table 9
Reference points.

Scenario	Type	Reference point
Scenario 2D	Restricted	(0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0)
	Compromised	(0.0, 0.0, 0.5, 0.5, 0.5, 0.5, 0.5)
Scenario 3D	Restricted	(1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0)
	Compromised	(0.5, 0.0, 0.5, 0.0, 0.0, 0.5, 0.5)

Table 10
Parameter settings.

Parameter	Algorithm	Value
Population size	All	112
Max evaluations	All	134,400
Crossover operator	All	Single Point Crossover
Crossover probability	All	0.9
Mutation operator	All	Bit Flip Crossover
Mutation probability	All	0.005
ϵ	R-NSGA-II	0.001
# of items for evaluation	COR-NSGA-II	5
# of reductions	COR-NSGA-II	10

5.5. Parameter settings

We adopted the type and values for crossover and mutation probabilities as defined in related work [12]. Then, we adopt the same probability rates for all algorithms, being 90% for crossover probability and 0.5% for mutation one. Regarding the population size and the maximum number of evaluations (considered as a stopping criterion), a tuning phase was performed and we tested two settings for these values: 112 and 238 for population size and 134,400 (or each solution is going to be evaluated 1200 times) and 238k (or each solution is going to be evaluated 1000 times) for a maximum number of evaluations. The population size was defined based on the NSGA-III mechanism. The latter uses a set of reference points on a hyper-plan where, by using p divisions for each objective, the number of reference points (and consequently the population size) is calculated by $H = \binom{M+p-1}{p}$, where M is the number of objectives [18].

Other specific parameters were also tuned. For example, ϵ for R-NSGA-II, was evaluated with 0.0001 and 0.001 (the same in [12]) where this one controls the number of solutions inside the ROI. Concerning COR-NSGA-II, the minimum confidence level (80% and 100%), the number of items for evaluation (5 and 10 items) and the number of reductions (5 and 10 ones) were also evaluated. In addition to this, R-Metric requires a δ, a parameter that specifies the ROI's size. For this work, the value of 0.3 was used. This value is the same used in the previous work for the addressed problem [12].

It is also important to notice that for **RQ1** and **RQ2**, COR-NSGA-II divides the maximum number of evaluations by the number of reductions. It means COR-NSGA-II is going to perform the same number of evaluations of other algorithms no matter the number of reductions. For example, suppose we have 900 as a maximum number of evaluations and COR-NSGA-II performs 3 reductions where each one runs the optimization algorithm for 300 evaluations.

Thus, 30 independent runs were performed using the combination of the parameters. After tuning, the best parameter settings were selected based on the best average values of R-HV and R-IGD. At the end, the values chosen are displayed in Table 10.

With the best configuration of parameters chosen, 30 independent runs of each algorithm were performed for answering **RQ1** and **RQ2**. At the end, the set of non-repeated and non-dominated solutions was obtained. As a statistical test, Kruskal–Wallis [56] with 95% significance level was considered where the bold values in the tables represent the best ones, and light gray cells represent values that are statistically equivalent. Finally, the algorithms were executed in a machine with an Intel(R) Core(TM) i7-5930K CPU 3.50 GHz with 40Gb RAM.

6. Results

In this section we present and analyze the results in order to answer our RQs.

6.1. RQ1: Sanity check

This RQ seeks to compare the results found by COR-NSGA-II to those found by a random dimensionality reduction algorithm (or simply, a random algorithm). The results found by COR-NSGA-II with 80% and 100% of minimum confidence level are presented, and concerning the random algorithm, the results are shown with 5 and 10 reductions. To ease understanding, we present the results of both experiments (Scenarios 2D e 3D) in separated tables.

Related to Scenario 2D, Table 11 shows the mean values and standard deviations for each column for COR-NSGA-II and random algorithm for Scenario 2D (*Number of Product* and *Alive Mutants* objectives as preferred ones).

Table 11 shows COR-NSGA-II reaches the best performance in all instances in which it can converge to the target objectives in 100% of the executions. On the contrary, the random algorithm always reaches just one objective in the last execution, and in most cases, this one is not a preferred objective. So, we can conclude that the random algorithm was the worst for this scenario.

Regarding the minimum confidence level used by COR-NSGA-II, the performance was similar (statistically equivalent) for most instances, except for James and CAS. In these ones, the COR-NSGA-II algorithm with 80% of minimum confidence level reached the best results, converging to the preferred objective in fewer reductions.

Then, we can observe that the sanity check has passed (i.e., COR-NSGA-II outperforms the random algorithm by a large degree). Besides, we can assume that the confidence level of 80% is better for this scenario once it slightly scores a better performance compared to 100%, by converging towards the preferred objectives quickly.

Table 12 shows the results for COR-NSGA-II and random algorithm regarding Scenario 3D (*Alive Mutants*, *Similarity*, and *Cost* objectives as preferred ones). COR-NSGA-II reaches the best performance in all instances in which it can converge to the preferred objectives in 100% of the executions. The random algorithm obtained a performance similar to that one obtained in Scenario 2D. It presented the worst performance by reducing the set to just one objective and this, in most cases, was not the preferred objective. Concerning the minimum confidence level used by COR-NSGA-II in this scenario, the performance was also similar (statistically equivalent) for most instances, except again for James and CAS. In these instances, COR-NSGA-II with 80% of confidence level converged to the preferred objective with fewer reductions.

Summarizing the results found in this scenario, we can observe that the sanity check has also passed. Besides, we can also assume (such Scenario 2D) that the minimum confidence level of 80% is better for the addressed problem once it slightly scores a better performance compared to 100% (by converging to preferred objectives quickly, in general, when compared to the other).

Key findings: COR-NSGA-II passes the sanity check. It is capable of reducing the problem dimensionality towards the user preferences in all cases for both scenarios.

6.2. RQ2: Comparing COR-NSGA-II to MOEAs and MaOEAs

To answer RQ2, Table 13 shows the number of times that the algorithms generated the best results for each quality indicator considering all instances. However, the comprehensive results can be found in our supplementary material [44].

Table 11
COR-NSGA-II vs. random algorithm in Scenario 2D.

	Algorithm	Reduction efficiency	Size of the last subset	# of Targets in the Last Subset	Reduction capacity
James	random-10	0.00 ± 0.00	1.00 ± 0.00	0.27 ± 0.45	0.00% ± 0.00
	random-5	0.00 ± 0.00	1.03 ± 0.18	0.40 ± 0.50	0.00% ± 0.00
	cor-nsga-ii-0.8	**1.70 ± 0.47**	**2.00 ± 0.00**	**2.00 ± 0.00**	**100.00% ± 0.00**
	cor-nsga-ii-1.0	2.03 ± 0.32	**2.00 ± 0.00**	**2.00 ± 0.00**	**100.00% ± 0.00**
CAS	random-10	0.00 ± 0.00	1.00 ± 0.00	0.27 ± 0.45	0.00% ± 0.00
	random-5	0.00 ± 0.00	1.00 ± 0.00	0.33 ± 0.48	0.00% ± 0.00
	cor-nsga-ii-0.8	**1.53 ± 0.51**	**2.00 ± 0.00**	**2.00 ± 0.00**	**100.00% ± 0.00**
	cor-nsga-ii-1.0	1.63 ± 0.49	**2.00 ± 0.00**	**2.00 ± 0.00**	**100.00% ± 0.00**
WS	random-10	0.00 ± 0.0	1.00 ± 0.00	0.27 ± 0.45	0.00% ± 0.00
	random-5	0.00 ± 0.00	1.07 ± 0.25	0.27 ± 0.45	0.00% ± 0.00
	cor-nsga-ii-0.8	**1.00 ± 0.00**	**2.00 ± 0.00**	**2.00 ± 0.00**	**100.00% ± 0.00**
	cor-nsga-ii-1.0	**1.00 ± 0.00**	**2.00 ± 0.00**	**2.00 ± 0.00**	**100.00% ± 0.00**
E-Shop	random-10	0.00 ± 0.00	1.00 ± 0.00	0.13 ± 0.35	0.00% ± 0.00
	random-5	0.00 ± 0.00	1.03 ± 0.18	0.40 ± 0.50	0.00% ± 0.00
	cor-nsga-ii-0.8	**1.00 ± 0.00**	**2.00 ± 0.00**	**2.00 ± 0.00**	**100.00% ± 0.00**
	cor-nsga-ii-1.0	**1.00 ± 0.00**	**2.00 ± 0.00**	**2.00 ± 0.00**	**100.00% ± 0.00**
Drupal	random-10	0.00 ± 0.00	1.00 ± 0.00	0.40 ± 0.50	0.00% ± 0.00
	random-5	0.00 ± 0.00	1.00 ± 0.00	0.20 ± 0.41	0.00% ± 0.00
	cor-nsga-ii-0.8	**1.00 ± 0.00**	**2.00 ± 0.00**	**2.00 ± 0.00**	**100.00% ± 0.00**
	cor-nsga-ii-1.0	**1.00 ± 0.00**	**2.00 ± 0.00**	**2.00 ± 0.00**	**100.00% ± 0.00**
Smarthome	random-10	0.00 ± 0.00	1.00 ± 0.00	0.27 ± 0.45	0.00% ± 0.00
	random-5	0.00 ± 0.00	1.00 ± 0.00	0.47 ± 0.51	0.00% ± 0.00
	cor-nsga-ii-0.8	**1.00 ± 0.00**	**2.00 ± 0.00**	**2.00 ± 0.00**	**100.00% ± 0.00**
	cor-nsga-ii-1.0	**1.00 ± 0.00**	**2.00 ± 0.00**	**2.00 ± 0.00**	**100.00% ± 0.00**

Table 12
COR-NSGA-II vs. random algorithm in Scenario 3D.

	Algorithm	Reduction efficiency	Size of the last subset	# of Targets in the Last Subset	Reduction capacity
James	random-10	0.00 ± 0.00	1.00 ± 0.00	0.53 ± 0.51	0.00% ± 0.00
	random-5	0.00 ± 0.00	1.00 ± 0.00	0.40 ± 0.50	0.00% ± 0.00
	cor-nsga-ii-0.8	**1.83 ± 0.53**	**3.00 ± 0.00**	**3.00 ± 0.00**	**100.00% ± 0.00**
	cor-nsga-ii-1.0	2.07 ± 0.25	**3.00 ± 0.00**	**3.00 ± 0.00**	**100.00% ± 0.00**
CAS	random-10	0.00 ± 0.00	1.00 ± 0.00	0.43 ± 0.50	0.00% ± 0.00
	random-5	0.00 ± 0.00	1.00 ± 0.00	0.47 ± 0.51	0.00% ± 0.00
	cor-nsga-ii-0.8	**1.20 ± 0.41**	**3.00 ± 0.00**	**3.00 ± 0.00**	**100.00% ± 0.00**
	cor-nsga-ii-1.0	1.33 ± 0.48	**3.00 ± 0.00**	**3.00 ± 0.00**	**100.00% ± 0.00**
WS	random-10	0.00 ± 0.00	1.00 ± 0.00	0.43 ± 0.50	0.00% ± 0.00
	random-5	0.00 ± 0.00	1.00 ± 0.00	0.40 ± 0.50	0.00% ± 0.00
	cor-nsga-ii-0.8	**1.00 ± 0.00**	**3.00 ± 0.00**	**3.00 ± 0.00**	**100.00% ± 0.00**
	cor-nsga-ii-1.0	**1.00 ± 0.00**	**3.00 ± 0.00**	**3.00 ± 0.00**	**100.00% ± 0.00**
E-Shop	random-10	0.00 ± 0.00	1.00 ± 0.00	0.37 ± 0.49	0.00% ± 0.00
	random-5	0.00 ± 0.00	1.00 ± 0.00	0.43 ± 0.50	0.00% ± 0.00
	cor-nsga-ii-0.8	**1.00 ± 0.00**	**3.00 ± 0.00**	**3.00 ± 0.00**	**100.00% ± 0.00**
	cor-nsga-ii-1.0	**1.00 ± 0.00**	**3.00 ± 0.00**	**3.00 ± 0.00**	**100.00% ± 0.00**
Drupal	random-10	0.00 ± 0.00	1.00 ± 0.00	0.37 ± 0.49	0.00% ± 0.00
	random-5	0.00 ± 0.00	1.00 ± 0.00	0.57 ± 0.50	0.00% ± 0.00
	cor-nsga-ii-0.8	**1.00 ± 0.00**	**3.00 ± 0.00**	**3.00 ± 0.00**	**100.00% ± 0.00**
	cor-nsga-ii-1.0	**1.00 ± 0.00**	**3.00 ± 0.00**	**3.00 ± 0.00**	**100.00% ± 0.00**
Smarthome	random-10	0.00 ± 0.00	1.00 ± 0.00	0.33 ± 0.48	0.00% ± 0.00
	random-5	0.00 ± 0.00	1.07 ± 0.25	0.50 ± 0.57	0.00% ± 0.00
	cor-nsga-ii-0.8	**1.00 ± 0.00**	**3.00 ± 0.00**	**3.00 ± 0.00**	**100.00% ± 0.00**
	cor-nsga-ii-1.0	**1.00 ± 0.00**	**3.00 ± 0.00**	**3.00 ± 0.00**	**100.00% ± 0.00**

Regarding COR-NSGA-II, NSGA-II and NSGA-III, Table 13 shows that COR-NSGA-II generates the best results for all scenarios and most quality indicators. Just for Scenario 3D and the largest instances, the performance of COR-NSGA-II are slightly worst, but it takes a reduced time to execute and generates a lower number of solutions in the ROI. The table also presents that COR-NSGA-II can outperform the R-NSGA-II algorithm in all scenarios. Specifically for Scenario 3D and the largest instances, the results found by COR-NSGA-II is slightly worst (similar to

Table 13
COR-NSGA-II vs. MOEAs and MaOEAs.

	Algorithm	R-HV	R-IGD	# of Sol.	# of Sol. in the ROI	Execution time (ms)
2D	COR-NSGA-II	**7**	**9**	**12**	**12**	**12**
	NSGA-II	4	3	0	0	0
	NSGA-III	1	4	0	0	0
3D	COR-NSGA-II	**8**	**7**	**6**	**10**	**12**
	NSGA-II	4	5	0	2	0
	NSGA-III	0	0	**6**	0	0
	Algorithm	R-HV	R-IGD	# of Sol.	# of Sol. in the ROI	Execution time (ms)
2D	COR-NSGA-II	**8**	**10**	**12**	**12**	**12**
	R-NSGA-II	4	4	0	0	0
3D	COR-NSGA-II	**8**	**8**	**12**	**12**	**12**
	R-NSGA-II	4	4	0	0	0
	Algorithm	R-HV	R-IGD	# of Sol.	# of Sol. in the ROI	Execution time (ms)
2D	COR-NSGA-II	**9**	**11**	**12**	**11**	**12**
	PCA-NSGA-II	3	3	0	1	0
3D	COR-NSGA-II	**8**	**8**	**10**	**10**	**12**
	PCA-NSGA-II	4	4	2	2	0

those found when it compared to NSGA-II and NSGA-III). However, it maintains spending less Execution Time (around 5 h on average lesser than R-NSGA-II), and generating a lower number of solutions in the ROI (around 100 solutions on average lesser than R-NSGA-II). More details in [44].

Finally, Table 13 describes that COR-NSGA-II reaches the best values for all scenarios when compared to PCA-NSGA-II. Considering the Execution Time, it is important to notice that the proposed algorithm reaches the lowest execution time where, in some cases, this one can be less than half of the time spent by other algorithms.

Key findings: COR-NSGA-II reaches, in general, fewer solutions in the ROI and these ones have a good performance when compared to those found by the other algorithms. On the one hand, COR-NSGA-II outperforms the other algorithms in most instances. On the other hand, its performance is slightly decreased when the Compromised RP and Scenario 3D with redundant objectives (not necessarily together) are considered.

6.3. RQ3: Evaluating The solutions

RQ3 evaluates the usefulness of the COR-NSGA-II solutions according to the user preferences compared to those found MOEAs and MaOEAs. For answering this RQ, the users were required to run Nautilus [21] and pick a solution up from the population generated by COR-NSGA-II, R-NSGA-II, and NSGA-II (the best algorithms found in **RQ2**). Also, they were required to, manually, provide a solution (Nautilus also supports this process).

Before performing the experiment, the users were required to select at most 4 objectives (out of 7) as preferred ones. This was required seeking to guarantee COR-NSGA-II will perform some reduction. Fig. 13 shows the preferred objectives from the user's point of view.

On the one hand, 11 users (91% of them) selected *Cost* as the preferred objective, followed by *Number of Products* selected by 10 users (or 83.3%). On the other hand, *Unselected Features* and *Uncovered Pairs* were less preferred by the users. Details about the set of preferred objectives selected by the participants and the final set of objectives generated by COR-NSGA-II are shown in [44].

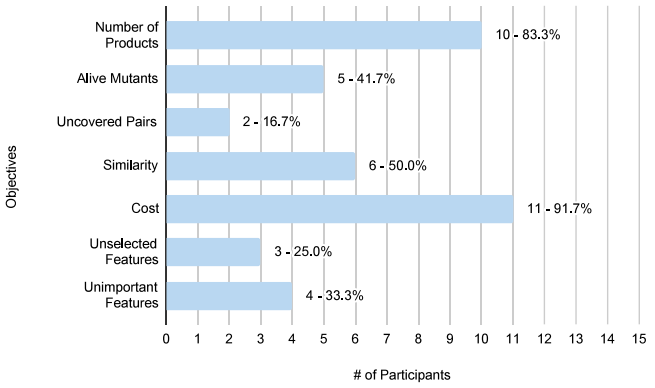


Fig. 13. Preferred objectives from the user's point of view.

Table 14
COR-NSGA-II's results for each participant.

Part.	# of Preferred Objectives	# of Reductions	Size of the Last Subset	# of targets in the Last Subset
#1	2	3	3	2
#2	2	2	2	2
#3	3	2	3	3
#4	2	4	2	2
#5	4	1	4	4
#6	4	1	3	3
#7	4	1	3	3
#8	4	1	5	4
#9	4	1	5	4
#10	4	1	6	4
#11	4	2	5	4
#12	4	2	4	4

Table 14 presents detailed information for each participant about the # of Preferred Objectives, # of Reductions performed by COR-NSGA-II, Size of the Last Subset (or the # of objectives optimized during the last algorithm execution), and # of Targets in the Last Subset (or # preferred objectives during the last algorithm execution).

Table 14 shows that 3 users selected 2 objectives, 1 user selected 3 objective as preferred, and 8 ones selected 4 objectives. The participants #2, #3, #4, #5, and #12 selected a solution when in the last execution, the set of objectives optimized contained just the preferred ones (Size and # of Targets in the Last Subset are the same). However, the participants #1, #8, #9, #10, and #11 were able to pick a solution up before COR-NSGA-II reaches this convergence.

After interacting with Nautilus and selecting a good solution for each algorithm, we asked the users in a post-survey to describe how difficult was the selection of this solution. Fig. 14 shows the feedback captured from the questionnaire. This figure describes that 8 users have chosen the option "Easy" and "Very Easy" for COR-NSGA-II, 4 for the NSGA-II and R-NSGA-II algorithms. In the last position, the manual selection appears as the most difficult (8 users). Analyzing the motivation, some participants claimed COR-NSGA-II generated fewer solutions and other ones claimed COR-NSGA-II took less execution time compared to the other algorithms. With this, we asked the users to rank the algorithm based on his/her preferences where 1 means the best one and so on. The information is displayed in Fig. 15.

Fig. 15 shows that COR-NSGA-II was ranked as the best algorithm by 10 users (or around 83%). NSGA-II was ranked as the best 2 times. As expected, the manual selection received the lowest score in this experiment, being ranked in the last position by 9 users (around 80%). However, aiming to verify the dependence on the execution order, Table 15 shows the best algorithm by group.

Two users from Group 4 selected the NSGA-II algorithm as the best one. In such a group, COR-NSGA-II was the first algorithm evaluated. We suppose that the results found by COR-NSGA-II influenced the

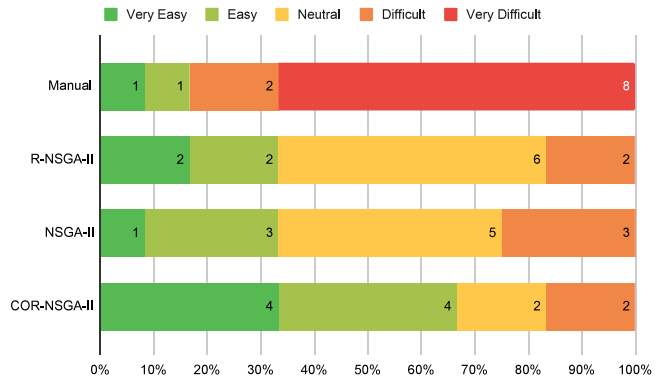


Fig. 14. Easiest algorithms from the user's point of view.

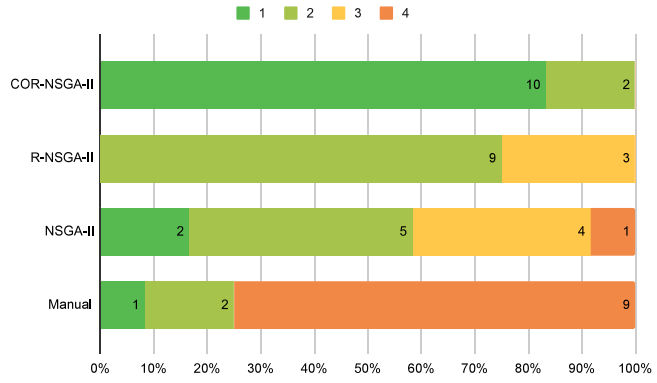


Fig. 15. Best algorithms from the user's point of view.

Table 15
Best algorithm by group.

Group 1	Group 2	Group 3	Group 4
COR-NSGA-II	COR-NSGA-II	COR-NSGA-II	COR-NSGA-II NSGA-II

user decision about the solutions generated by the other algorithms, i.e., they used the solution selected by applying COR-NSGA-II to guide his/her preferences in the following algorithms and this can be a motivation for the other algorithms appear as good as COR-NSGA-II. However, more experiments must be performed to validate our hypothesis. A fact that can corroborate this hypothesis is that we asked the users if the execution order in each group (described in Table 8) impacted their responses. Around 58.3% answered positively.

Key findings: COR-NSGA-II can help the user to find useful solutions for the addressed problem since 10 out of 12 users defined that the solutions found by COR-NSGA-II were the best ones compared to other algorithms and it was also easy to select them.

7. Implications and future work

As presented in the previous section, COR-NSGA-II can generate fewer solutions, taking less execution time for searching them, but maintaining the quality attributes in a competitive way. However, in this section, we discuss some implications of our experimental study results.

As expected in **RQ1**, the random dimensionality reduction algorithm was the worst one for the addressed problem once it does not take into account the user preferences. In all cases, this one ends the search process simply because it reached the minimum number of objectives (in case, a single one). On the contrary, COR-NSGA-II reached the preferred objectives, on average, in the second reduction. As in this experiment a user simulator was used and this one is responsible for selecting randomly solutions in the Pareto-front, the number of reductions could be less if a kind of smart mechanism were used in this task. Also, the execution time they take to execute should be evaluated in future experiments.

In **RQ2**, in most cases, COR-NSGA-II found better results for the addressed problems. However, it is possible to notice a slightly decreasing of the values regarding R-HV and R-IGD when the number of preferred objectives increases. We suppose that the presence of redundant objectives in this set can affect the algorithm performance. In addition to this, we also suppose that if the set of preferred objectives has almost the same size as the original set of objectives, the performance must not be competitive since COR-NSGA-II may not perform multiple reductions. New experiments should be performed in the future to verify this assumption with different scenarios. Although this can happen, the difference between COR-NSGA-II and the other algorithms remains small and it is a good trade-off. For example, considering R-NSGA-II in Scenario 3D, the performance of COR-NSGA-II was around 3% worse than the former for the Smarthome instance (the largest one). However, for the same context, R-NSGA-II took 8 h on average to find a solution, while COR-NSGA-II, with its dimensionality reduction mechanism, took 3.2 h on average, i.e., a reduction in the execution time of more than 50%.

This is another important finding of our experiment. COR-NSGA-II took less time in all instances and compared with algorithms, and the difference among the algorithms increases when the instance size increases as well. This becomes COR-NSGA-II an important algorithm in the context of where the number of products to be selected is huge. However, future experiments might confirm these results specially because this RQ uses a simulated user. Also, the performance of COR-NSGA-II remains good when the Compromised RP is considered. Different from Restricted RP, the former considers 0.5 in the RP for the non-preferred objectives (that is, although a small value, it still is a user preference for this objective). However, COR-NSGA-II does not take into account this concept, that is, for the algorithm, if an objective to be optimized is not preferred, it is simply discarded. So, more research in the future should evaluate the performance of the proposed algorithm in the context where there are some small preferences for non-preferred objectives and some random reference points were provided. Moreover, the use of COR-NSGA-II is recommended in the context where all objectives should be optimized, but some of them are preferred. If, in the user's point of view, all objectives have the same preferences, traditional MOEAs and MaOEAs should be applied. In addition to this, future experiments should be performed to verify, in case all objectives should be removed, if to find the closest value to $minConf$ could perform better.

Finally, concerning to **RQ3**, two unexpected situations were identified during the performance of the experiments. The first situation occurs when a user selected two redundant objectives as preferred. Because of this, the manual mechanism for selecting a solution was considered by this user better than the optimization algorithms used in this experiment (although the user took more than 20 min to pick a solution up in using the manual mechanism). Another finding was that some users took more than 10 min exploring the Pareto-front, providing his/her preferences. Since the experiment did not set a limit for the users to express his/her preferences in COR-NSGA-II, some users spent time providing his/her preferences as much as they can. Thus, seeking to improve this process, it is necessary in a future work, to study a kind of limit for the number of preferences and, also, to propose a way to deal with redundant objectives in the set of preferred ones. Future work should also apply some techniques for data visualization or usability in order to improve the selection of user preferences.

8. Threats to validity

Internal Validity. Regarding RPs used in **RQ2**, the results are dependent on the reference points used. These ones were selected considering our previous knowledge about the Pareto-front and the set of preferred objectives. So, aiming to mitigate this threat, two kinds of RPs were considered in which the non-preferred objectives in the first RP do not have any preferences, while for the second one compromised values (0.5 to be exact) are provided to non-preferred objectives. To better evaluate the impact of the provided RPs, we need to conduct future experiments, even with random RPs. Furthermore, the experiments with users showed that the algorithm is highly dependent on the context in which it is applied and the set of preferred objectives defined by the user, especially because the participants do not work in the system on which the FMs were based. Hence, all users received the same information and, before performing the tests, all of them took part in a preliminary test to learn how to use the tool, and prevent any effects of ignorance on their usage. We use the tool FMTS to calculate the objective values of our problem. FMTS makes use of the FaMa framework to deal with resource model constraints and derive products. However, FaMa has some limitations to work with large FMs such as Drupal and Smarthome. To mitigate this threat, the user can set a number n of products to be used for selection. Other representations for the problem can be used in the future, as well as other analyzers and tools for setting "n" more accurately, such as SAT solver and Glencoe [57], respectively. The randomly selected "n" products could also be a threat to validation. To mitigate this in our experiments, we considered "n" values that are proportional to the number of products and, at the end, the mutation score value was around 97% for each FM, i.e., 3% of mutants were discarded considering the percentage found for the smaller FMs determined using the complete set of products.

Construct Validity. We used questionnaires to assess the comprehension of the solution selection process and the participants' answers to these questionnaires were evaluated by comparing the answers with the quality metrics for them. This design choice avoided as much as possible any subjective evaluation. Thus, the questionnaires were defined to be complex enough without being too obvious.

External Validity. We tested COR-NSGA-II in six different instances of SPL Testing. Even though these instances are evaluated in other studies of the literature, we cannot state that this is enough to generalize the results. Besides, the cost and importance (both of them were randomly defined), and the size of the instances may not reflect real-world FMs. To minimize this threat, we tried to evaluate FMs of several sizes (including two with more the 11k products to be selected) and domains. Regarding the larger instances, we consider that 11k of products to be selected were adequate proportionally to the mutation score (around 97%) chosen for the experiments performed in this work. However, a greater number for large SPLs should be evaluated in a future experiment. It is expected to have a similar performance and good results with acceptable time. Regarding the definition of the scenarios, the selection of the objective should be a threat. Aiming to minimize it, we tried to select them based on redundancy or conflicting nature.

Conclusion Validity. The parameter settings used by the algorithms can be a threat. The number of evaluations is the same for everyone, even COR-NSGA-II that applies dimensionality reduction. Different values, especially in the larger instances, could result in different, perhaps better, capacity of reducing towards user preferences. To address the stochastic nature of the evolutionary algorithms, all the algorithms were performed 30 times for each instance and reference points to answer **RQ1** and **RQ2**, while capturing the arithmetic mean and standard deviation of the metrics. Also, Kruskal–Wallis test with 95% significance level was used to compare the results found by the algorithms. This test is quite robust and it has been extensively used in the past to conduct similar analyses.

9. Concluding remarks

This work presented COR-NSGA-II, an algorithm that reduces the number of objectives to be optimized towards the user needs based on a confidence level for each objective optimized. As main characteristics, COR-NSGA-II is an algorithm that requires the user preferences interactively (or in-the-loop) and reduces the number of objectives during the solution generation process. The algorithm shows to the user the current set of non-dominated solutions and, at this point, the user can express his/her preferences about them by using an ordinal scale. For assessing the feasibility of COR-NSGA-II, multiple experiments were performed using six different FMs, two types of reference points, five algorithms, and two scenarios.

We compared COR-NSGA-II to a random dimensionality objective algorithm and concluded that the proposed algorithm is, in fact, capable of guiding the search process to the objectives preferred by the users. Also, the results found by COR-NSGA-II were compared to those found by MOEAs and MaOEAs. By using several quality indicators, we observed that COR-NSGA-II, in most instances and scenarios, obtained the best results or results statistically equivalent to the algorithms evaluated, even when the Compromised RP is considered. In this sense, we can conclude that COR-NSGA-II, indeed, generates a small set of good solutions taking less time to execute and, mainly, incorporating the user preferences. Besides, a group of users was invited to optimize a problem instance and, the great majority of the users, answered that it is easier to pick a solution up generated by COR-NSGA-II, and chose this algorithm as the best compared with the other ones.

Hence, considering the evaluation done in this work and the answers found for the research questions, we can conclude that COR-NSGA-II is capable of quickly generating a small set of solutions that satisfy the user preferences, still keeping the solutions as good as those generated by MOEAs and MaOEAs used in the VTSPL literature. Future work include (i) to perform more empirical study to evaluate the scalability of COR-NSGA-II in other FMs used in the industry; (ii) to evaluate COR-NSGA-II in an environment in which the number of reductions is limited; (iii) to study new values for confidence level for removing an objective and increase the feedback options used by COR-NSGA-II; (iv) to apply COR-NSGA-II to other SE problems; and (v) to extend COR-NSGA-II with other optimization algorithms as search engine.

CRediT authorship contribution statement

Thiago Ferreira: Conceptualization, Methodology, Formal analysis, Investigation, Data curation, Software, Writing – original draft, Formal analysis, Validation, Writing – review & editing. **Silvia Regina Vergilio:** Conceptualization, Data curation, Validation, Investigation, Methodology, Writing – review & editing, Supervision, Project administration, Funding acquisition. **Marouane Kessentini:** Validation, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared the link to our data/code in the paper

Acknowledgments

This work is supported by CAPES, Brazil and CNPq, Brazil, grants: 307762/2015-7 and 305968/2018-1.

References

[1] M.B. Cohen, M.B. Dwyer, J. Shi, Coverage and adequacy in software product line testing, in: Proceedings of the 15th International Symposium on Software Testing and Analysis (ISSTA' 06), ACM, Portland, USA, 2006, pp. 53–63.

[2] M. Harman, B.F. Jones, Search-based software engineering, Inf. Softw. Technol. 43 (2001) 833–839.

[3] S. Wang, S. Ali, A. Gotlieb, Minimizing test suites in software product lines using weight-based genetic algorithms, in: Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '13), ACM, Amsterdam, The Netherlands, 2013, pp. 1493–1500.

[4] S. Wang, D. Buchmann, S. Ali, A. Gotlieb, D. Pradhan, M. Liaaen, Multi-objective test prioritization in software product line testing: An industrial case study, in: Proceedings of the 18th International Software Product Line Conference (SPLC '14), ACM, Florence, Italy, 2014, pp. 32–41.

[5] F. Ensan, E. Bagheri, D. Gašević, Evolutionary search-based test generation for software product line feature models, in: Proceedings of the 25th International Conference on Advanced Information Systems Engineering (CAiSE '13), Springer, Valencia, Spain, 2012, pp. 613–628.

[6] C. Henard, M. Papadakis, G. Perrouin, J. Klein, Y.L. Traon, Multi-objective test generation for software product lines, in: Proceedings of the 17th International Software Product Line Conference (SPLC '13), ACM, Tokyo, Japan, 2013, pp. 62–71.

[7] C. Henard, M. Papadakis, Y. Le Traon, Mutation-based generation of software product line test configurations, in: Proceedings of the 6th International Symposium on Search Based Software Engineering (SSBSE '14), Springer, Fortaleza, Brazil, 2014, pp. 92–106.

[8] R.E. Lopez-Herrejon, F. Chicano, J. Ferrer, A. Egyed, E. Alba, Multi-objective optimal test suite computation for software product line pairwise testing, in: Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM '13), IEEE, Eindhoven, The Netherlands, 2013, pp. 404–407.

[9] R.A. Matnei-Filho, S.R. Vergilio, A mutation and multi-objective test data generation approach for feature testing of software product lines, in: Proceedings of the 29th Brazilian Symposium on Software Engineering (SBES'15), IEEE Computer Society, Belo Horizonte, Brazil, 2015, pp. 21–30.

[10] A. Strickler, J.A. Prado Lima, S.R. Vergilio, A. Pozo, Deriving products for variability test of feature models with a hyper-heuristic approach, Appl. Soft Comput. 49 (2016) 1232–1242.

[11] T. do Nascimento Ferreira, J.A.P. Lima, A. Strickler, J.N. Kuk, S.R. Vergilio, A. Pozo, Hyper-heuristic based product selection for software product line testing, IEEE Comput. Intell. Mag. 12 (2) (2017) 34–45.

[12] H.L. Jakubovski-Filho, T. do Nascimento Ferreira, S.R. Vergilio, Preference based multi-objective algorithms applied to the variability testing of software product lines, J. Syst. Softw. 151 (2018) 194–209.

[13] T. do Nascimento Ferreira, S.R. Vergilio, M. Kessentini, Applying many-objective algorithms to the variability test of software product lines, in: Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing (SAST '20), Natal, Brazil, 2020, pp. 11–20.

[14] J. Parejo, A. Sánchez, S. Segura, A. Ruiz-Cortés, R. Lopez-Herrejon, A. Egyed, Multi-objective test case prioritization in highly configurable systems: A case study, J. Syst. Softw. 122 (2016) 287–310.

[15] M.W. Mkaouer, M. Kessentini, S. Bechikh, K. Deb, M. Ó Cinnéide, High dimensional search-based software engineering: Finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III, in: Proceedings of the 16th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '14), ACM, Vancouver, Canada, 2014, pp. 1263–1270.

[16] A.S. Sayyad, H. Ammar, Pareto-Optimal search-based software engineering (POSBSE): A literature survey, in: Proceedings of the 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '03), IEEE, San Francisco, USA, 2013, pp. 21–27.

[17] T.E. Colanzi, W.K. Assunção, S.R. Vergilio, P.R. Farah, G. Guizzo, The symposium on search-based software engineering: Past, present and future, Inf. Softw. Technol. 127 (2020) 106372.

[18] K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints, IEEE Trans. Evol. Comput. 18 (4) (2014) 577–601.

[19] B. Li, J. Li, K. Tang, X. Yao, Many-objective evolutionary algorithms: A survey, ACM Comput. Surv. 48 (1) (2015) 13.

[20] K. Deb, D. Saxena, Searching for Pareto-optimal solutions through dimensionality reduction for certain large-dimensional multi-objective optimization problems, in: Proceedings of the 8th IEEE Congress on Evolutionary Computation (CEC '06), Vancouver, Canada, 2006, pp. 3352–3360.

[21] T. do Nascimento Ferreira, S.R. Vergilio, M. Kessentini, Nautilus: An interactive plug and play search based software engineering framework, IEEE Softw. (2020).

[22] F.J. Van der Linden, K. Schmid, E. Rommes, Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering, Springer Science & Business Media, 2007.

[23] G. Perrouin, S. Sen, J. Klein, B. Baudry, Y. Le Traon, Automated and scalable t-wise test case generation strategies for software product lines, in: Proceedings of the 3rd IEEE International Conference on Software Testing, Verification and Validation (ICST '10), IEEE, Paris, France, 2010, pp. 459–468.

[24] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, Y. Le Traon, Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines, IEEE Trans. Softw. Eng. 40 (7) (2014) 650–670.

[25] S. Oster, M. Zink, M. Lochau, M. Grechanik, Pairwise feature-interaction testing for SPLs: Potentials and limitations, in: Proceedings of the 15th International Software Product Line Conference (SPLC '11), ACM, Munich, Germany, 2011, p. 6.

[26] E. Uzuncaova, S. Khurshid, D. Batory, Incremental test generation for software product lines, IEEE Trans. Softw. Eng. 36 (3) (2010) 309–322.

[27] D.M. Cohen, S.R. Dalal, J. Parelius, G.C. Patton, The combinatorial design approach to automatic test generation, IEEE Softw. 13 (5) (1996) 83–88.

[28] D. Reuling, J. Bürdek, S. Rotärmel, M. Lochau, U. Kelter, Fault-based product-line testing: Effective sample generation based on feature-diagram mutation, in: Proceedings of the 19th International Software Product Line Conference (SPLC '15), ACM, Nashville, Tennessee, 2015, pp. 131–140.

[29] P. Arcaini, A. Gargantini, P. Vavassori, Generating tests for detecting faults in feature models, in: Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST '15), IEEE, Graz, Austria, 2015, pp. 1–10.

[30] J.M. Ferreira, S.R. Vergilio, M.A. Quináia, A mutation approach to feature testing of software product lines, in: Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering (SEKE'13), Knowledge Systems Institute Graduate School, Boston, USA, 2013, pp. 232–237.

[31] C. Henard, M. Papadakis, G. Perrouin, J. Klein, Y. Le Traon, Assessing software product line testing via model-based mutation: An application to similarity testing, in: Proceedings of the 6th IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW '13), IEEE, Luxembourg, 2013, pp. 188–197.

[32] K. Deb, Multi-Objective Optimization using Evolutionary Algorithms, Vol. 16, John Wiley & Sons, 2001.

[33] E.K. Burke, G. Kendall, Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer, 2005.

[34] A. Zhou, B. Qu, H. Li, S. Zhao, P.N. Suganthan, Q. Zhang, Multiobjective evolutionary algorithms: A survey of the state of the art, Swarm Evol. Comput. 1 (1) (2011) 32–49.

[35] R. Akbari, R. Hedayatzadeh, K. Ziarati, B. Hassanizadeh, A multi-objective artificial bee colony algorithm, Swarm Evol. Comput. 2 (2012) 39–52.

[36] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.

[37] C. Audet, J. Bigeon, D. Cartier, S. Le Digabel, L. Salomon, Performance indicators in multiobjective optimization, European J. Oper. Res. 292 (2) (2021) 397–422.

[38] H. Jain, K. Deb, An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: Handling constraints and extending to an adaptive approach, IEEE Trans. Evol. Comput. 18 (4) (2014) 602–622.

[39] H. Abdi, L.J. Williams, Principal component analysis, Wiley Interdiscip. Rev. Comput. Stat. 2 (4) (2010) 433–459.

[40] K. Deb, J. Sundar, U. Bhaskara, S. Chaudhuri, Reference point based multi-objective optimization using evolutionary algorithms, Int. J. Comput. Intell. Res. 2 (3) (2006) 27–286.

[41] C.H.P. Kim, D.S. Batory, S. Khurshid, Reducing combinatorics in testing product lines, in: Proceedings of the Tenth International Conference on Aspect-Oriented Software Development, 2011, pp. 57–68.

[42] C.H.P. Kim, D. Marinov, S. Khurshid, D. Batory, S. Souto, P. Barros, M. d'Amorim, SPLat: Lightweight dynamic analysis for reducing combinatorics in testing configurable systems, in: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, 2013, pp. 257–267.

[43] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, A. Jimenez, Fama framework, in: Proceedings of the 12th International Software Product Line Conference (SPLC '08), IEEE, Limerick, Ireland, 2008, p. 359.

[44] T. do N. Ferreira, S.R. Vergilio, M. Kessentini, Supplementary material, 2021, https://nautilus-framework.github.io/cor-nsga-ii/.

[45] T. do Nascimento Ferreira, J.N. Kuk, A. Pozo, S.R. Vergilio, Product selection based on upper confidence bound MOEA/D-DRA for testing software product lines, in: Proceedings of the 18th IEEE Congress on Evolutionary Computation (CEC '16), IEEE, Vancouver, Canada, 2016, pp. 4135–4142.

[46] H.L. Jakubovski-Filho, T. do Nascimento Ferreira, S.R. Vergilio, Multiple objective test set selection for software product line testing: Evaluating different preference-based algorithms, in: Proceedings of the XXXII Brazilian Symposium on Software Engineering (SBES '18), ACM, Sao Carlos, Brazil, 2018, pp. 162–171.

[47] D. Benavides, S. Trujillo, P. Trinidad, On the modularization of feature models, in: Proceedings of the 1st European Workshop on Model Transformation (CMT '06), Bilbao, Spain, 2005, p. 134.

[48] S. Weißleder, D. Sokenou, B. Schlingloff, Reusing state machines for automatic test generation in product lines, in: Proceedings of the 1st Workshop on Model-Based Testing in Practice (MoTiP '08), Berlin, Germany, 2008, pp. 19–28.

[49] D. Beuche, M. Dalgarno, Software product line engineering with feature models, Overload J. 78 (2007) 5–8.

[50] K.C. Kang, J. Lee, P. Donohoe, Feature-oriented project line engineering, IEEE Softw. 19 (4) (2002) 58–65.
[51] A.A. Araújo, M. Paixão, Machine learning for user modeling in an interactive genetic algorithm for the next release problem, in: Proceedings of the 6th International Symposium on Search Based Software Engineering (SSBSE '14), Springer, Fortaleza, Brazil, 2014, pp. 228–233.
[52] M. Shackelford, D.W. Corne, A Technique for Evaluation of Interactive Evolutionary Systems, Springer, 2004, pp. 197–208.
[53] T. do Nascimento Ferreira, A.A. Araújo, A.D. Basílio-Neto, J.T. de Souza, Incorporating user preferences in ant colony optimization for the next release problem, Appl. Soft Comput. 49 (2016) 1283–1296.
[54] E. Zitzler, Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications, (Ph.D. thesis), Citeseer, 1999.
[55] K. Li, K. Deb, X. Yao, R-metric: Evaluating the performance of preference-based evolutionary multiobjective optimization using reference points, IEEE Trans. Evol. Comput. 22 (6) (2017) 821–835.
[56] R. Kuhn, R. Kacker, Y. Lei, J. Hunter, Combinatorial software testing, Computer 42 (8) (2009) 94–96.
[57] R. Heradio, D. Fernandez-Amoros, J.A. Galindo, D. Benavides, Uniform and scalable SAT-sampling for configurable systems, in: Proceedings of the 24th ACM Conference on Systems and Software Product Line (SPLC '20), ACM, Montreal, Canada, 2020, pp. 1–11.