Preference Based Multi-Objective Algorithms Applied to the Variability Testing of Software Product Lines

Helson Luiz Jakubovski Filho[a], Thiago Nascimento Ferreira[a,*], Silvia Regina Vergilio[a]

[a]*DInf - Federal University of Paraná, CP: 19097, CEP: 81.531-980, Curitiba, Brazil*

Abstract

Evolutionary Multi-Objective Algorithms (EMOAs) have been applied to derive products for the variability testing of Software Product Lines (SPLs), which is a complex task impacted by many factors, such as the number of products to be tested, coverage criteria, and efficacy to reveal faults. But such algorithms generally produce a lot of solutions that are uninteresting to the tester. This happens because traditional search algorithms do not take into consideration the user preferences. To ease the selection of the best solutions and avoid effort generating uninteresting solutions, this work introduces an approach that applies Preference-Based Evolutionary Multi-objective Algorithms (PEMOAs) to solve the problem. The approach is multi-objective, working with the number of products to be tested, pairwise coverage and mutation score. It incorporates the preferences before the evolution process and uses the Reference Point (RP) method. Two PEMOAs are evaluated: R-NSGA-II and r-NSGA-II, using two different formulations of objectives, and three kinds of RPs. PEMOAs outperform the traditional NSGA-II by generating a greater number of solutions in the Region of Interest (ROI) associated to the RPs. The use of PEMOAs can reduce the tester's burden in the task of selecting a better and reduced set of products for SPL testing.

Keywords: Software Product Line Testing, Search-Based Software Engineering, Preference-Based Algorithms

*Corresponding author

Email addresses: `hljfilho@inf.ufpr.br` (Helson Luiz Jakubovski Filho), `tnferreira@inf.ufpr.br` (Thiago Nascimento Ferreira), `silvia@inf.ufpr.br` (Silvia Regina Vergilio)

1. Introduction

The variability testing of Software Product Lines (SPLs) is devoted to test a set of products derived from the SPL variability model. The most popular and used variability model is the Feature Model (FM) [1], considered as the facto standard in the SPL engineering. Ideally, all the products should be tested, but the number of possible products grows exponentially with the number of features and the exhaustive testing is, in general, infeasible [2]. Then, the use of a test criterion can help in the selection of the best products to be tested. A test criterion is a predicate to be satisfied by the test data set and is generally used to consider that the testing activity has ended [3].

In the FM context, the test cases are SPL products and these ones need to satisfy some criteria. The most popular criteria [2,4–7] are the ones derived from combinatorial testing [8]. Among them, the pairwise testing is the most used [2,4,7,9,10], which requires the selection of products to include all valid pairs of features present in the FM. The number of covered pairs is used to evaluate the set of products generated.

More recent works are based on mutation testing [11–15]. Mutant FMs are generated to describe typical faults that can be present in the FM. The idea is to find, for each mutant, a product that is validated by the mutant and is not validated by the original FM [11], or otherwise, it is valid for the FM but it is invalid for the mutant. When a difference exists the mutant is considered killed. At the end, the mutation score is produced and used to evaluate a set of products according to the number of killed mutants.

The selection of products for SPL testing can be impacted by different factors such as: criterion coverage; cost (given by the number of products); efficacy (related to the number of faults found); and so on. Different solutions can be considered as good alternatives, according to the testing goals and the testers' needs. Due to this, such a problem has been efficiently solved by Evolutionary Multi-Objective Algorithms (EMOAs), in the field known as Search-Based Software Engineering (SBSE) [16]. Surveys on search-based SPL engineering [17,18] shows that there are many works devoted to this research subject [19–21]. The most used algorithms are Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [22], Strength Pareto Evolutionary Algorithm (SPEA2) [23], and Indicator-Based Evolutionary Algorithm (IBEA) [24]. Such algorithms provide a set called non-dominated solutions, distributed over the Pareto-front. At the end, all these solutions are considered Pareto-equivalent and good solutions. The Decision Maker (DM) is then responsible for choosing the solution to be used.

However, this choice is not always easy for the DM, because the cardinality of the non-dominated solution set is, most times, very large, especially if the number of objectives becomes high. In our problem, that is, product selection for SPL testing, it is not true that all the solutions are equivalent from the DM's point of view. For instance, the DM, (the tester in our case) is not usually interested in solutions with low coverage. Therefore, it is clear that the objectives are not equally important for the DM.

Cases like that are very common in SBSE, and to cope with these limitations, a new field has emerged, named Preference and Search-Based Software Engineering (PSBSE) [25]. PSBSE is the SBSE sub-research field devoted to the application of preference and search-based algorithms to solve software engineering problems. A preference-based algorithm is the one that incorporates human preferences, intuition, emotion or psychological aspects in the optimization process [26]. There are different ways to incorporate the user preferences [25,27]: before (a priori), during (interactively), or after (a posteriori) the solution evolution process.

The best way depends on several conditions related to the DM's personality, context, problem's characteristics, and so on. For example, consider our SPL testing problem, in many cases there is a coverage level for the system under test or a cost budget, which are known a priori. In this way, it is suitable an a priori approach instead of an interactively one that requires a lot of effort and can increase the DM's burden. It is important to notice that a recent mapping on PSBSE [25] does not mention this kind of approach applied in the SPL context.

Considering the above-mentioned facts, in this paper we introduce a

Preference-Based Evolutionary Multi-objective Algorithm (PEMOA) approach to select products for the variability test of SPLs. The goal is to find the best solutions according to the tester's needs, and considering trade-offs related to some factors such as cost, efficacy and criteria coverage. The proposed approach has two main characteristics:

- It is a mutiobjective approach. This allows the use of multiple factors that impact our problem. Ferreira et al. [25] point out that most solutions in PSBSE are mono-objective. Besides, the authors consider the use of multi-objective approach is a research gap in PSBSE;
- It applies two PEMOAs based on the well-known and used algorithm NSGA-II. They are: Reference Point-based NSGA-II (R-NSGA-II) [28] and Reference Solution-based NSGA-II (r-NSGA-II) [29]. Both algorithms are based on the Reference Point (RP) concept. For this problem, the tester is interested only in a subset of solutions of the Pareto front, and the preferred part of the Pareto optimal region from the DM's point of view is called Region of Interest (ROI). Then, the DM's preference guides the PEMOA in the search, which results in a greater number of solutions in the ROI, close to the RP. This eases the task of choosing the final solution to be used.

These characteristics present some advantages: i) the approach contributes to the PSBSE field and can be used to solve other similar Software Engineering problems; ii) the preferences can be provided a priori, requiring reduced effort from the DM; iii) the existence of a ROI facilitates the visualization of the solutions (in the case of problems with two or three objectives) and recognizing if they satisfy the DM's preferences; and iv) the approach can provide a greater number of solutions in the ROI space, reducing the number of uninteresting solutions and, consequently, easing the decision making task.

The paper also presents results that compare both PEMOAs, considering formulations to the problem with two and three objectives. The results show

PEMOAs can generate better solutions than NSGA-II by avoiding uninteresting solutions (outside the ROI space). In general, R-NSGA-II presents the best performance and generates a greater number of solutions inside the ROI.

The paper is organized as follows. Section 2 reviews main PSBSE concepts and the PEMOAs used in this work. Section 3 contains related work. Section 4 introduces the proposed approach: problem formulation, search operators and objective functions. Section 5 describes how the empirical evaluation was conducted: research questions, quality indicators, used FMs, and algorithms configuration. Section 6 presents and analyses the results. Section 7 discusses the main threats to the validity of our results. Finally, Section 8 concludes the work and presents future research directions.

2. Preference and Search-Based Software Engineering

As mentioned before, the Preference and Search-Based Software Engineering (PSBSE) field is the SBSE sub-field devoted to the application of preference-based algorithms to solve software engineering problems [25]. A preference-based algorithm is the one that incorporates human preferences, intuition, emotion or psychological aspects in the optimization process [26]. To this end, the Decision Maker (DM) plays a key role, since the DM is the person (or a group of persons) who provides the human preferences. It is believed that the DM has the best insights about the problem being the best person who can express the preference relations between different solutions [27]. In the context of this work, the DM is the tester.

The process of embedding user preferences can occur at different stages of the algorithm execution. Usually, these stages take into account three main basic incorporation moments: i) before (a priori): the user must specify his/her preference information before the execution of the algorithm. For example, the user provides their preferences and these are represented as an objective or translated to weights; ii) after (a posteriori): the user must specify his/her preference information after the execution of the algorithm, such as providing a way to rank the solutions; iii) during (interactively): the user preferences are provided during the execution of the algorithm. For example, during the execution of the algorithm, the user can interactively provide a number representing his/her preferences that impacts the search.

However beyond the moment to incorporate the user preference in the optimization process, it is necessary to define how this information will be provided, referred to the incorporation method [25,30]. The method defines, for example, what information the DM visualizes, what information is required from the DM, how the preferences influence in the search, and so on. The best way to incorporate preferences depends on several conditions such as the user's personality, the context, the characteristics of the problem, and so on.

One method very popular and most used that is suitable to a priori approach is the Reference Point (RP) method [31]. The use of a RP (also called the aspiration level vector) corresponds to points in the search space, where the DM would like the objectives to be concentrated. In this way, the RP is presented as a natural way of expressing DM preferences.

This method performs the projection of the RP reported by the DM over the optimal Pareto region via the minimization of an Achievement Scalarizing Function (ASF). The use of RPs can guide the search toward a Region of Interest (ROI) without demanding effort from the tester, even if the number of objectives increases. This can not happen when using an interactive approach. In this way, the ROI is composed as a set of non-dominated solutions that are preferred by the DM [32]. These solutions are co-located within the area of the Pareto front on which the DM would like to focus and that usually contains the RP. Figure 1 shows an example of the ROI's representation.

Based on the Pareto front, Said et al. [29] offer a classification for possible points that can be provided by the DM. They are illustrated in Figure 2 and can be: (A) a small distance from (or can belong to) the Pareto front; (B) feasible, a point that can be reached and possibly generated as solution by an algorithm; and (C) infeasible, a point that can not be reached or obtained as solution.

In this work, we apply two Preference-Based Evolutionary Multi-objective

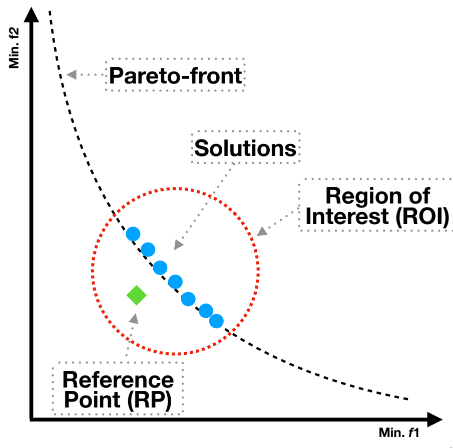

Figure 1: ROI's Representation (adapted from [33]).

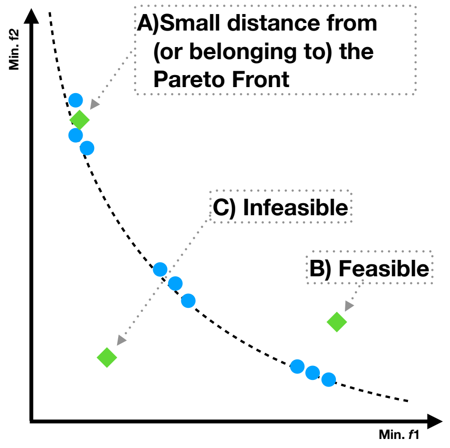

Figure 2: Classification of reference point method (adapted from [29]).

Algorithms (PEMOAs), which use the RP (or aspiration vector) and are based on the NSGA-II algorithm [22]. NSGA-II is a strong elitist algorithm that in each generation creates a new set of solutions (the offspring) based on their parents, joins the offspring with the parents and sorts them according to their dominance rank and crowding distance. Therefore, at each generation, the solutions that are non-dominated when compared to the other ones survive to the next generation. A solution x is said to dominate another solution y ($x \prec y$)

if it is better or equal to y in all objectives, and is better in at least one objective. If this is not true, then x and y are non-dominated. When there are several non-dominated solutions and the next generation can not receive all of them, 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. Next we describe the PEMOAs used in this work.

2.1. Reference Point-Based NSGA-II

Proposed by Deb et al. [28] and based on NSGA-II, the Reference Point-based NSGA-II (R-NSGA-II) algorithm aims to guide the ROI search according to DM preferences, by means of RP method.

The algorithm operation is similar to NSGA-II, but differs in some points. First, the DM must provide one or more RPs for the algorithm. Second, the crowding distance metric is modified, being called "preferred distance" because it represents how the solutions are close to the RPs, that is, the use of this metric implies in giving a greater emphasis to the solutions that are closer to the RP provided by the DM. Finally, 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, emphasizes the closest solutions to the RP.

The preferred distance metric implemented by R-NSGA-II works 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 in the top of the rank; 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 [28].

The execution of the ϵ-clearing procedure occurs as follows: i) all solutions having a sum of normalized difference in objective values of ϵ or less between them are grouped; ii) a solution is randomly chosen from each group; iii) for the other members of the group is attributed a great crowding distance in order to discourage them to remain in the race. Next, another solution is chosen from the set of non-dominated solutions (excluding the one previously chosen) and the procedure is performed again. The value of ϵ is chosen according to the application, thus it consists of a parameter informed by DM [28].

2.2. Reference Solution-Based NSGA-II

The Reference Solution-based NSGA-II (r-NSGA-II) was introduced by Said et al. [29]. This algorithm uses a variant of the Pareto dominance relation called r-dominance, derived from the hybridization between the Pareto dominance principle and the RP method.

The main idea behind the concept of r-dominance is to define a relation between Pareto-equivalent solutions. Thus, the r-dominance has the ability to differentiate non-dominated solutions in a partial way based on a vector of aspiration levels (RPs) provided by the DM. This fact not only makes the r-dominance selection pressure stronger than the Pareto dominance one but also integrates the DM's preferences in the selection process [29].

The algorithm r-NSGA-II has some additional parameters. The parameter δ allows the DM to control the selection pressure of the r-dominance relation. When $\delta = 1$, the relation of r-dominance is equal to Pareto dominance, if $\delta = 0$, only the non-dominated solutions closest to the RP are emphasized [29]. With these characteristics, the δ parameter can be used to ensure population diversity during the evolutionary process.

To that end, Said et al. [29] use a form of adaptive management of the r-non-dominance threshold δ during the evolution process, functioning as follows: Assuming that n_ge is the number of generations, which establishes the stopping criterion, i is the index of the current generation of the evolutionary cycle and δ_0 is the value of δ provided by the DM, in the initialization stage ($i = 0$), the r-NSGA-II algorithm classifies the population based on Pareto dominance

($\delta = 1$). From the first generation, the value of δ is truncated in each generation, according to the relation $1 - i * ((1 - \delta)/n\_ge$. Thus, in the last generation, the value of δ corresponds to the value initially reported by the DM (δ_0). The objective of this process is to guide the search gradually during the evolution process to determine the ROI, so that it is possible to avoid the phenomenon of premature convergence [29].

The algorithm NSGA-II is then executed until the stopping criterion is met but using the r-dominance instead of the traditional Pareto dominance.

3. Related Work

Search-based algorithms have been applied to solve different problems in the SPL engineering [17,18], including variability testing of SPLs. We can find works on prioritization and selection of test cases. Both tasks are viewed as complementary. The work of Parejo et al. [34] proposes a multi-objective approach for prioritization of SPL products in order to accelerate fault-detection. Existing works proposing selection approaches are the most related to ours. They use different factors (objectives) and algorithms (mono and multi-objective ones). The most used objectives are: number of products in the selected set, number of revealed faults, the similarity of products, pairwise coverage, and mutation score. The approaches that use mono-objective algorithms [35–39], in many cases, use aggregation functions. The most commonly used algorithms are (1+1) Evolutionary Algorithm (EA), Random Search, and Genetic Algorithm (GA).

Wang et al. [35] used a weighted GA to minimize SPL test suites while seeking to retain fault detecting power. To this end, they defined three effectiveness measures: test minimization percentage, feature pairwise coverage and fault detecting capability. After this, in another publication [36], the authors introduced a test prioritization approach that considers cost and effectiveness measures. In this context, the proposal was evaluated comparing GA with (1+1) EA and random search.

Ensan et al. [37] search for SPL feature interactions by using GA with an aggregation function comprised of cost and error rate factors. Henard et al. [38] use a GA to handle multiple conflicting objectives: costs, pairwise coverage, and the number of products in SPL test data generation.

In the mutation testing context, Henard et al. [39] introduce a search-based approach which explores the SPL product space to generate test cases (products) using mutation operators defined to generate dissimilar products with the aim of detecting mutants. For this, the approach implements the (1+1) EA algorithm in conjunction with a constraint solver to check if the products are valid.

Lopez-Herrejon et al. [19] formulate a bi-objective problem approach which considers pairwise coverage and the size of the test suites. The works of Matnei-Filho and Vergilio [20,21] propose a multi-objective approach and use three objectives: number of test cases, pairwise coverage, and mutation score. The authors evaluate different EMOAs: NSGA-II, SPEA2 and IBEA. As a result the authors conclude that there is no significant difference between the algorithms evaluated, and the NSGA-II algorithm performed slightly better for the general case. Other multi-objective approaches explore the use of hyper-heuristics with multi-objective algorithms [40–42]. The use of hyper-heuristics can help in the construction of more generic solutions and ease the implementation of the algorithms.

Among the mentioned works, multi-objective approaches [19–21,40,41] are the most promising, since the product selection for SPL testing is, in fact, a multi-objective problem impacted by many factors. We also can observe that NSGA-II obtained good results. On the other hand, the use of a PEMOA to solve this problem was not explored by the above-mentioned works. A recent mapping on PSBSE [25] corroborates to this fact. In general, the use of PEMOAs has been few explored for software engineering problems and, also, most of the works found in the mapping use mono-objective algorithms.

However, the mapping mentions some initiatives in the software testing area and SPL engineering that are related to our work, despite they do not address the same problem. In the software testing area the works of Marculescu et al. [43] and Feldt [44] describe interactive systems for deriving test data. In both approaches, the tester participates in the evolutionary process providing his/her preferences to improve the quality of selected test cases. The work of Kalboussi et al. [45] is more related to ours, since it uses a PEMOA for testing of autonomous agents. The algorithm r-NSGA-II is used in a many-objective approach and the RP used to guide the search for hard test cases, that is, test cases not related to soft goals of the agent.

In the SPL context, the work of Yamany et al. [46] introduces an interactive search-based approach to configure SPLs. The approach is implemented by a tool named Opti-Select, which runs a loop allowing the DM to select, according to his/her preferences, the best solutions in the Pareto front. These solutions impact the next step of the loop, which continues until the desired solution is obtained. An improved version of this tool, the Smart Opti-Select, was proposed in [47]. Such a version implements a machine learning module to learn the DM's preferences by combining results from the execution of NSGA-II and IBEA.

We can see that there is no work that applies a PEMOA in the SPL context. In fact, we find only one preference-based approach to configure SPLs, and such approach is interactive, considers the user-in-the-loop. This kind of approach presents some disadvantages: it can cause fatigue and requires a lot of effort. Considering this aspect and the characteristics of our problem, we describe in the next section our approach that is a priori and based on the RP method.

4. Proposed approach

The proposed approach has as the main goal to generate solutions that best satisfy the preferences expressed by the DM. Each solution represents a set of products that optimizes different factors that impact the variability testing of SPLs.

Variability testing is responsible for ensuring that the products derived from the variability model match expected requirements. Our approach is based on the Feature Model (FM), which is largely adopted in the industry.

Figure 3 shows an example of FM for a Mobile Phone SPL that will be used in this section to illustrate the approach. Ideally, all possible products derived from the FM should be tested. However, this is not always possible, because the number of products can grow exponentially to the number of features [2]. The use of a testing criterion can help the selection of most representative products.

A testing criterion can also provide a way to select and evaluate the test data, which in this context are the products to be tested. Then, our problem is to determine a minimum set of products that satisfy a given testing criterion. Such a set is supposed to have a great capacity to reveal faults, which can be present in the FM, with reduced (possible optimal) cost.

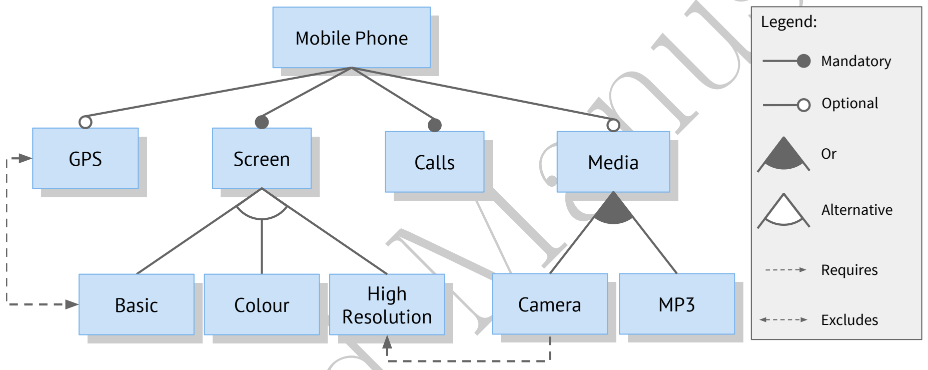

Figure 3: FM of a mobile phone SPL [42].

According to Ferreira et al. [25] to develop a PSBSE approach, four main ingredients are necessary: i) a representation for the problem, which allows the search algorithm to manipulate the solutions; ii) the set of manipulation operators; iii) a fitness function to evaluate the quality of the solutions; and iv) a way to incorporate the user preferences. Such ingredients of our approach are described in this section.

4.1. Population Representation

An individual (or solution) in our population consists of a binary encoding, where each gene represents a product derived for a given FM under test. When the i-th bit is equal to 1 the product i belongs to the solution. Otherwise, the i-th bit is equal to 0. The same convention is used to represent the features selected for a product. The value (1) represents that the corresponding feature is selected for the product, and (0) represents that the feature is not selected. Figure 4 shows an example of an individual for the FM of Figure 3. In the example, the chromosome representation uses four products ($n = 4$) for the FM. The individual S, represented in the figure, includes the products p_2 and p_3. Then, the number of selected products of S is $n_s = 2$. The product p_2 in the right side of the figure is represented in terms of its variabilities, and contains the features `High Resolution` and `Camera`.

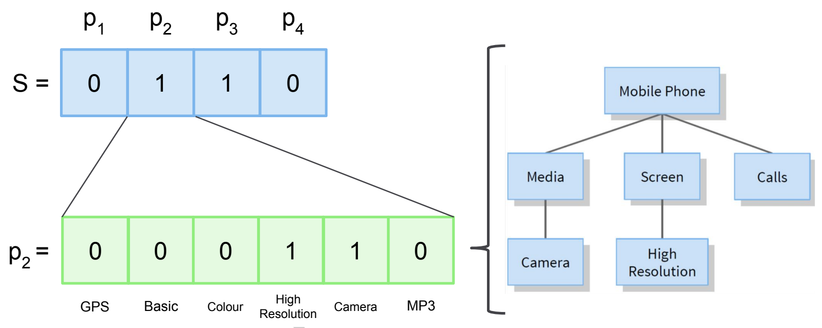

Figure 4: Individual representation.

4.2. Search Operators

In our approach, we adopted the following search operators: uniform crossover, bit flip mutation and binary tournament for selection of the individuals.

In the application of the uniform crossover, each descending gene is created by copying the corresponding gene from one of the chosen parents, according to a randomly generated crossover mask. If 1 is generated, the corresponding gene will be copied from the first parent and if 0 is generated, it will be copied from the second.

To apply the bit flip mutation, a random position in the gene is selected, if this position contains 0 it is changed to 1, and so vice versa.

Regarding the binary tournament selection, two individuals are randomly chosen from the population to be the parents based on the dominance principle, which classifies individual solutions into different dominance levels. If there is no dominance among them, one is randomly chosen.

4.3. Objective Functions

The individuals are evaluated according to the main objectives used in the literature [21,40,41], related to the size of the test data set, mutation score and pairwise coverage. All of them are described as follows.

Let S be an individual generated by an algorithm with n_s selected products. The first objective (Equation 1) is related to the cost factor, and corresponds to the total number of products in S. This number is expressed as the ratio of the number of selected products (n_s) and the number of valid products being considered for the FM (n). It is computed as following:

$$T(S) = \frac{n_s}{n} \tag{1}$$

In our work, the number of products n is given by the framework Feature Model Analyser (FaMa) [48]. It supports the FODA notation [1], extended and cardinality-based FMs. However, for huge FMs, the tester can provide a desired value for n (see section 4.5).

The second objective (Equation 2) is related to the efficacy, that is, the capacity of the product set to reveal faults, that in our case, is given by the mutation score with respect to a set of mutation operators that represent possible faults that can be present in an FM. The objective is defined as:

$$A(S) = 1 - \frac{KM}{AM} \tag{2}$$

where KM is the number of killed mutants by the products in S, and AM is the total number of active mutants. We used the set of mutation operators and the FMTS tool proposed by Ferreira et al. [11,49]. This tool works with FaMa [48], which is responsible for validating the FM being tested and its mutants. Only valid mutants, which are validated by FaMa, are generated by FMTS. The goal is to generate a product that can be derived by a model and not in the other. To reach this goal, the product p is checked by the framework FaMa. The mutant is considered killed if p is valid according to the original FM and invalid for the mutant, or otherwise, p is invalid for the FM and valid for the mutant. When both FMs, original and mutant ones, validate the same set of products, they are considered equivalent. The set AM of active mutants is composed by valid and non-equivalent mutants.

Figure 5 presents an example of mutant for the FM of Figure 3. The mutation operator changes a requires relation to an excludes one, such as the one between `High Resolution` and `Camera`. In this way, a product that contains both features is valid for the original FM and invalid for the corresponding mutant, which will be considered killed. In this sense, the product p_2 in Figure 4 kills the mutant, since it is valid for the original FM and is invalid for the mutant.

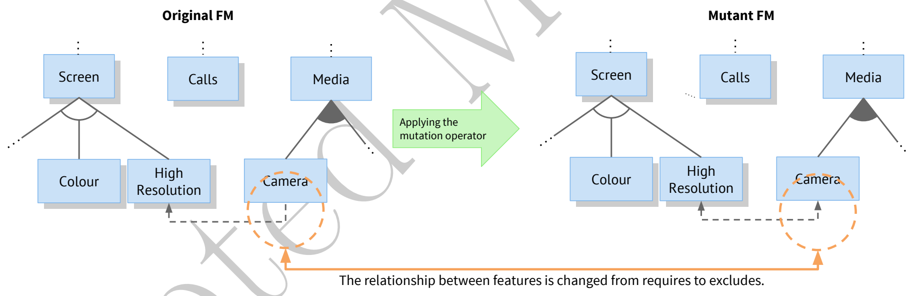
Original FM
Mutant FM
Screen
Calls
Media
Colour
High Resolution
Camera
Applying the mutation operator
The relationship between features is changed from requires to excludes.

Figure 5: Example of a mutant created from mobile phone [42].

Finally, the third objective (Equation 3) corresponds to the pairwise coverage, where CP represents the number of pairs covered by S and VP means the total number of valid pairs.

$$P(S) = 1 - \frac{CP}{VP} \tag{3}$$

For example, by analyzing Figure 3, the pair (`GPS`, `Basic`) is invalid, and should not be required. If we consider only the variabilities, we see that the product p_2 of Figure 4 covers the pair (`High Resolution`, `Camera`). In this work, we use the Combinatorial tool[1] to derive the pairs. This tool implements the Automatic Efficient Test Generator (AETG) algorithm [8].

4.4. Preference Incorporation Method

By analyzing the characteristics of our problem, we observe in many cases a test plan is available that contains the desired coverage for the testing criteria being used, as well as a limit to the number of products to be tested, due to cost restrictions.

In this way, even without any knowledge about the solution space, it is not difficult for the tester to specify values for the objectives to be evaluated. Due to this we chose the Reference Point (RP) method [31]. The use of other methods such as modeling preferences as weights or trade-offs between objectives, to be used in a priori approach seems a difficult and complicated task for our problem. In this way, we apply two PEMOAs: R-NSGA-II and r-NSGA-II, described in Section 2, which are based on the NSGA-II algorithm [22].

Such an algorithm is the most popular and used in the SBSE field [16], and has presented good results for SPL problems [17,18], including our selection problem [20,21] as mentioned in Section 3.

4.5. Implementation Aspects

To implement the algorithms we use the framework jMetal [50]. To reduce execution time during the evolution process and to avoid many fitness calculations, the implementation uses input matrices, M_A and M_P that maintain, respectively, the mutants killed and the covered pairs by a product. These matrices are generated in the initial phase and have n entries, where n can be either the number of products generated by FaMa for the FM, or a possible smaller number provided by the tester. In this last case, n valid products are generated at random and will be used in the chromosome representation (see

[1]http://161.67.140.42/CombTestWeb [8].

Section 4.1). The fitness of all products is calculated. This mechanism allows scalability, avoiding the manipulation of a huge number of products.

As mentioned before, to calculate the fitness of the solutions, we used the FMTS tool and its operators, and the AETG algorithm implemented in the Combinatorial tool. FMTS works with FaMa analyzer. In this way, the FM under test is provided, following FaMa, in XML format. An array of features present in the diagram is generated and used by the AETG algorithm to generate the pairs. After this, a procedure is executed to discard invalid pairs, considering the variabilities and restrictions of the FM. A module to check if a product covers a given pair was also implemented. By using FMTS a set of mutants is generated. FMTS only generated valid mutants according to FaMa.

To create an individual S in the initial population, we first set n_s, the number of products S contains, by randomly chosen a number in the interval [1:MAX], where MAX varies between 1 and n, according to a percentage α provided as parameter. After this, n_s positions are randomly chosen in the chromosome and set to 1.

In the evolution process, the search operators (mutation and crossover) are applied according to the rates provided by the tester.

5. Empirical Evaluation

The hypothesis of this work is that our approach is capable to automatically generate a better set of solutions considering the DM preferences, given by the RP provided by the tester, and avoiding uninteresting solutions. According to such goals, the experiment was guided by the following research questions:

RQ1: How do the results of the PEMOAs compare to the results obtained by the traditional NSGA-II and to a random selection?

RQ2: What is the best PEMOA? How do r-NSGA-II and R-NSGA-II compare?

In addition to this, we want to compare the performance of the used PEMOAs, considering different RPs and objectives. Then, to answer the research questions and to evaluate the use of the approach with different sets of objectives, we created two experiments. The first experiment was carried out considering a 2-objectives formulation related to the number of products and the mutation score, to find a solution S that minimizes Equation 4.

$$\underset{S}{\text{minimize}} \quad (T(S), A(S)) \tag{4}$$

The second experiment was carried out considering a 3-objectives formulation that also includes the pairwise coverage. The idea of this second experiment is to increase the search complexity, making it more difficult. The goal is to find S that minimizes Equation 5.

$$\underset{S}{\text{minimize}} \quad (T(S), A(S), P(S)) \tag{5}$$

In the following sub-sections we describe quality indicators, target FMs, RPs, and parameters used to compare the algorithms in both experiments [2].

5.1. Quality Indicators

The analysis was conducted by using sets and indicators from the multi-objective optimization area [51] that are relevant to the scope of this work: i) Hypervolume in conjunction with R-Metric (R-HV); ii) Euclidean Distance (ED); and iii) average number of solutions and percentage of solutions in the ROI. To obtain such indicators, three sets of solutions were generated.

- PF_{approx}: set of non-dominated solutions obtained by one algorithm execution;
- PF_{known}: set of non-dominated solutions of an algorithm obtained by the union of all the PF_{approx} from all the executions, removing the non-dominated and repeated solutions;

[2]The raw data regarding both experiments and described in this section can be found at https://gres-ufpr.github.io/pmoas-for-spl/

- PF_{true}: represents the Optimal Pareto Front to the problem. In our case this set is unknown. Due to this, and following the literature [40,41,52], this set was formed by all sets PF_{known} obtained from different algorithms by removing dominated solutions and repeated ones. The set PF_{true} is, in fact, an approximation to the real front.

5.1.1. Hypervolume with R-Metric (R-HV)

Traditional performance evaluation metrics do not take into account the RP informed by the DM during the evaluation process. 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) proposed by Li et al. [33]. It provides a way to adapt quality indicators, such as the hypervolume (HV), to quantitatively evaluate the performance of a PEMOA by using RP.

The general idea of R-metric is to pre-process the preferred solutions according to a multi-criterion decision-making approach before using a regular metric to evaluate the performance of the obtained solutions.

The first step to calculate R-HV is to filter the solutions by keeping only the non-dominated and no-repeated ones. In the second step, a representative point is identified, which reflects the general satisfaction of the solutions with respect to the RP. In the third step, the main objective is to have a fair comparison between different sets by trimming points that are outside the ROI. After this, in the fourth step, the trimmed points are transferred to a virtual position to be evaluated its proximity to the RP. Finally, the last step applies the quality indicator in the solutions processed by R-Metric. In our case, the Hypervolume (HV) quality indicator [51].

The objective of the 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. The R-HV was calculated considering the sets PF_{approx} generated by each algorithm. At the end, the average of the results obtained by calculating the R-HV in each set is returned. For this, the objectives values were normalized between

$[0.0; 1.01]$. Thus, higher values of R-HV present the best results, that is, results that contain a set of solutions that are closer to the RP and also contain a greater number of solutions with good diversity within the ROI.

5.1.2. Euclidean Distance (ED)

This quality indicator measures the distance between a solution and the ideal one in the solution space. Considering the situation in which the DM chooses only one solution from the non-dominated set, we analyze the solutions with the smallest ED in relation to the RP reported by his/her. Thus, the goal is to measure how close a particular solution is to the RP.

To calculate ED, the PF_{known} set of each algorithm was used, according to Equation 6 that describes how to calculate the value between two points $T = \{t_1, t_2, ..., t_n\}$ and $Q = \{q_1, q_2, ..., q_n\}$ in an n-dimensional Euclidean space.

$$ED = \sqrt{\sum_{i=1}^{n}(t_i - q_i)^2} \tag{6}$$

5.1.3. Average Number of solutions in the ROI

This metric evaluates how well an algorithm can generate solutions in the ROI. In our analysis, we take into account the percentage of solutions generated within the ROI regarding to the total number of generated solutions, according to Equation 7.

$$P_{roi} = \frac{|S_{ROI}|}{|Sol|} \times 100 \tag{7}$$

where S_{ROI} represents the set of solutions that are inside the ROI and Sol is the set formed by all solutions obtained by the algorithm. The metric evaluates the algorithm ability to obtain concentrated solutions that satisfy the DM's preferences.

In order to perform the calculation of both metrics, the average was calculated considering the sets PF_{approx} formed by each algorithm and for the determination of the ROI, we made use of R-Metric previously described.

Typically in SBSE, the researchers use non-parametric tests, taking into account the stochasticity of search-based algorithms. In this sense, to compare the algorithms, for each problem/instance, we used the Kruskal-Wallis statistical test [53] with 95% significance level.

5.2. Target Feature Models

We used six FMs already used in related work [20,34,40,41], five of them extracted from the SPLOT repository [54]. These FMs are: a) James [55]: SPL for collaborative web systems; b) CAS (Car Audio System)) [56], to manage automotive sound systems; c) WS (Weather Station) [57]: SPL for weather forecast systems; d) E-Shop, an E-commerce SPL [58]; e) Drupal, a modular open source web content management framework [34]; and f) SmartHome v2.2: SPL for a smart residential solution [38].

Table 1 shows information about each FM, such as number of products (n_t), number of used products n, active mutants (AM), valid pairs (VP), and number of features ($Features$). 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.

The AM set does not include equivalent mutants, i.e., mutants that could not be killed by valid products according to the FM under test. This ensure the generation of a set composed only by valid products. For small FMs, it was possible to determine the equivalent mutants by using FaMa. A mutant was considered equivalent if the set of products derived by the mutant and the set derived by the original FM are the same. We observe that for these FMs, around 3% of the generated mutants are equivalent. This percentage was used to set the value of n for the two larger FMs. The number of products n was chosen ensuring a mutation score around 97%. In this way, the mutants that could not be killed by any one of the generated products were discarded in the input matrix M_A. For both FMs, the set of products cover all the pairs in the matrix M_P.

Table 1: Characteristics of the FMs used in the experiment.

FM	n_t	n	AM	P	**Features**
James	68	68	106	75	14
CAS	450	450	227	183	21
WS	504	504	357	195	22
E-Shop	1152	1152	94	202	22
Drupal	≈2.09E9	11k	2194	1081	48
SmartHome	≈3.87E9	11k	2948	1710	60

5.3. Definition of the Reference Points (RP)

To apply our approach the tester provides the RP according to their needs, without any knowledge about the problem space or Pareto fronts. But in our evaluation we do not have real testers and coverage levels required for each SPL, then to simulate such behavior we used three kinds of RPs according to the classification of Said et al. [29] (Section 2). These points represent the kind of points that the tester can provide and we call them as Feasible, Infeasible, and True described as follows.

- Feasible: we suppose that the user has an idea of a solution that is far from the Pareto front and can still be improved;
- Infeasible: in this case, the user provides a solution that is also far from the front but can not be reached by the algorithms.
- True: the user provided a solution that is a small distance from the front, which is given, in our case, by the set PF_{true}.

As the classes of RP were established taking into account the distance from the Pareto front, to set the values of RP we considered the fronts generated by algorithm NSGA-II, considering a range for the criteria coverage of 95% to 99% and a mean number of products of the solutions found in the front corresponding to this desired coverage range. But in practice, it is not necessary to execute the algorithms to choose the RPs. Table 2 shows the RPs used in both experiments. The mutation score is represented in our tables in terms of percentage.

Table 2: Reference points.

Formulation	FM	Feasible	Infeasible	True
2-Objectives	James	(6; 95%)	(2; 98%)	(5; 97%)
	CAS	(8; 96%)	(3; 97%)	(7; 97%)
	WS	(11; 96%)	(3; 97%)	(10; 98%)
	E-Shop	(11; 95%)	(3; 97%)	(9; 98%)
	Drupal	(25; 97%)	(8; 99%)	(16; 98%)
	SmartHome	(25; 96%)	(7; 99%)	(16; 98%)
3-Objectives	James	(6; 98%; 98%)	(1; 98%; 98%)	(3; 98%; 98%)
	CAS	(8; 96%; 96%)	(3; 98%; 98%)	(5; 98%; 99%)
	WS	(12; 98%; 98%)	(3; 97%; 97%)	(8; 98%; 99%)
	E-Shop	(12; 98%; 98%)	(3; 99%; 99%)	(5; 97%; 98%)
	Drupal	(27; 97%; 97%)	(10; 99%; 99%)	(18; 98%; 99%)
	SmartHome	(28; 96%; 97%)	(8; 99%; 99%)	(18; 99%; 99%)

5.4. Parameters Setting

We performed a parameter tuning based on values, already defined in related work that applied NSGA-II for the same problem and FMs [20,40,41]. Then, we adopt the same probability rates for the three algorithms: NSGA-II, R-NSGA-II and r-NSGA-II, being 90% for crossover probability and 0.5% for mutation one.

Regarding to the population size and maximum number of evaluations (considered as a stopping criterion), we adopted the same values used in [40], with 60,000 fitness evaluations and a population size equals to 200. For the larger FMs, as they have near to 10 times more products to be selected, we empirically determined such values by increasing 10 times the number of evaluations used in the small instances. So, we adopted a 600k as number of evaluations and population size of 200. Therefore, To evaluate the runtime, the same number of evaluations was given for all algorithms.

Regarding the α parameter, used to generate the initial population, a tuning phase was performed taking into account the values 1, 0.1, 0.01, and 0.001. To select the best one, the average of the Hypervolume and the execution time were considered for 10 independent runs. So, the value $\alpha = 0.01$ to generate the initial population for all FMs was selected.

We also performed a tuning to the specific parameters of the PEMOAs, by evaluating the following different values for the parameters: ϵ (R-NSGA-II parameter) equals to 0.0001, 0.001, and 0.01; and δ (r-NSGA-II parameter) equals to 0.1, 0.03, and 0.05. For both algorithms, the weight w used to emphasize each goal in the vector of aspiration levels was the same, with $w = 0.5$. Ten independent runs of each algorithm were performed using each combination of parameters. Infeasible RPs were used, since they represent solutions that are harder to improve.

After tuning, the best parameter settings were selected based on the best average values of R-HV. For those averages that did not reach statistical significance through the Kruskal Wallis test [53], with a significance level of 95%, the setting with the smallest mean runtime was chosen. At the end, the values chosen for the δ and ϵ parameters were the same for all FMs and for both formulations, being respectively 0.3 and 0.001.

With the best configuration of parameters chosen, 30 independent runs of each algorithm were performed [51]. The solutions of the random approach, used in our comparison as a baseline, were obtained by taking the initial populations generated in those 30 runs. At the end, the set of non-repeated and non-dominated solutions was obtained.

The algorithms were executed in two machines: a) on an Intel(R) Core(TM) i7-5930K CPU 3.50GHz with 40Gb RAM for 2-objectives formulation, and b) on an Intel(R) Core(TM) i7-4930K CPU 3.40GHz with 62 Gb RAM for 3-objectives formulation.

6. Results and Analysis

In this section, we present the experimental results and analyze them aiming at answering the posed questions. To ease understanding, we present the results of both experiments (formulations) in separated sections.

6.1. 2-Objectives Formulation

Table 3 shows the mean values and standard deviations for R-HV indicator. The bold values represent the best ones, and light gray cells represent values that are statistically equivalent.

Table 3: R-HV for 2-objectives formulation.

FMRP		**NSGA-II**	**RANDOM**	**r-NSGA-II**	**R-NSGA-II**
James	Inf.	0.4309 (0.1525)	0.1084 (0.0000)	0.7312 (0.1169)	**0.9252 (0.01035)**
	Feasible	0.5289 (0.1523)	0.1122 (0.0000)	**0.8966 (0.0031)**	0.8965 (0.0101)
	True	0.7588 (0.1707)	0.1126 (0.0000)	0.8992 (0.0025)	**0.9288 (0.0104)**
CAS	Infeasible	0.9222 (0.0204)	0.5883 (0.0330)	0.9760 (0.0107)	**0.9794 (0.0057)**
	Feasible	0.9762 (0.016)	0.5881 (0.0330)	0.9494 (0.0074)	**0.9806 (0.0060)**
	True	0.9769 (0.0078)	0.5846 (0.0326)	0.9494 (0.0073)	**0.9803 (0.0041)**
WS	Infeasible	0.9621 (0.0082)	0.6318 (0.0211)	0.9146 (0.0252)	**0.9730 (0.0050)**
	Feasible	0.9774 (0.0050)	0.6292 (0.0209)	0.9453 (0.0047)	**0.9813 (0.0035)**
	True	0.9568 (0.0046)	0.6205 (0.0204)	0.9754 (0.0086)	**0.9812 (0.0023)**
E-Shop	Infeasible	0.9859 (0.0053)	0.8594 (0.0256)	0.9649 (0.0076)	**0.9873 (0.0042)**
	Feasible	0.9621 (0.0116)	0.8712 (0.0262)	0.9518 (0.0084)	**0.9829 (0.0087)**
	True	**0.9896 (0.0026)**	0.8471 (0.0251)	0.9755 (0.0030)	0.9894 (0.0021)
Drupal	Infeasible	0.9965 (0.0014)	0.8841 (0.0147)	0.9963 (0.0021)	**0.9975 (0.0017)**
	Feasible	0.9756 (0.0029)	0.9003 (0.0151)	0.9817 (0.0038)	**0.9957 (0.0027)**
	True	0.9863 (0.0027)	0.8920 (0.0150)	0.98700 (0.0026)	**0.9964 (0.0030)**
SmartHome	Infeasible	0.9971 (0.0010)	0.8904 (0.0157)	0.9920 (0.0024)	**0.9978 (0.0012)**
	Feasible	0.9851 (0.0029)	0.9161 (0.0166)	0.9800 (0.0042)	**0.9953 (0.0030)**
	True	0.9863 (0.0020)	0.8983 (0.0160)	0.9881 (0.0025)	**0.9971 (0.0021)**

The table shows better performance of RNSGA-II in most cases. Just for

E-Shop, NSGA-II reached the best performance but without statistical difference. R-NSGA-II outperforms r-NSGA-II in most instances independently of the adopted RPs (16 out of 18 cases) even when Drupal and SmartHome (the largest instance) are considered. r-NSGA-II obtained the best performance for James when feasible RP is used, but again without statistical difference. We can observe that the random selection obtained the worst R-HV values in all cases.

Considering the ED indicator, we selected the solution from the PF_{known} of each algorithm that is the closest to the RPs. Table 4 shows those solutions and the ED values obtained. The bold values represent the best ones.

The table shows that NSGA-II outperforms the other algorithms just for James with infeasible RP and WS with feasible and true RPs. However, for other instances, we can see a better performance of the PEMOAs. The r-NSGA-II reached the best performance in most instances (14 out of 18). R-NSGA-II is better or equivalent than NSGA-II in 11 (out of 18) instances. The random selection reached a good ED value for SmartHome and feasible RP, but is worse than the other algorithms in most cases.

Another analysis considers Table 5 that shows the mean number of solutions in the ROI and the rate between the number of found solutions in the ROI and the total number of found solutions.

The table shows the random algorithm can generate the smallest number of solutions in most instances (14 out of 18). However, most of them are out of the ROI (in some cases, around 20% are inside the ROI) and when the number of products to be selected increases, the percentage tends to decrease (see SmartHome instance). The r-NSGA-II algorithm generates, for all cases, the smallest number of solutions and all of them (100%) are inside the ROI.

It is possible to see that NSGA-II and R-NSGA-II generates a similar number of solutions inside the ROI, but NSGA-II generates a greater number of solutions outside the ROI. On average 65% of the NSGA-II solutions are inside the ROI against around 80% of the R-NSGA-II solutions. The number of products in the FMs does not seem to impact these findings (the R-NSGA-II performance

Table 4: ED for 2-objectives formulation.

FMRP		NSGA-II	RANDOM	r-NSGA-II	R-NSGA-II
James	Infeasible	**0.1975 (3; 78.30)**	0.6594 (1; 32.07)	**0.1975 (3; 78.30)**	0.2070 (3; 77.35)
	Feasible	0.0500 (6; 100.0)	0.6335 (1; 32.07)	0.0500 (6; 100.0)	**0.0294 (5; 92.45)**
	True	0.0334 (6; 100.0)	0.6519 (1; 32.07)	**0.0111 (5; 98.11)**	0.0226 (6; 95.28)
CAS	Infeasible	0.1154 (4; 85.46)	0.1683 (4; 80.17)	**0.0539 (5; 91.63)**	0.1154 (4; 85.46)
	Feasible	0.0401 (9; 100.0)	0.1585 (4; 80.17)	**0.0312 (8; 99.11)**	0.0401 (9; 100.0)
	True	0.0303 (9; 100.0)	0.1684 (4; 80.17)	**0.0080 (7; 97.79)**	0.0186 (6; 95.15)
WS	Infeasible	0.1325 (5; 83.75)	0.2193 (4; 75.07)	**0.0934 (6; 87.67)**	0.1410 (5; 82.91)
	Feasible	**0.0205 (10; 98.03)**	0.1537 (5; 80.67)	0.0261 (10; 98.59)	0.0373 (12; 99.72)
	True	**0.0004 (10; 98.03)**	0.1736 (5; 80.67)	0.0088 (10; 98.88)	0.0063 (11; 98.59)
E-Shop	Infeasible	0.1121 (5; 85.78)	0.1654 (5; 80.45)	**0.0163 (8; 95.43)**	0.0995 (5; 87.05)
	Feasible	0.0501 (14; 100.0)	0.0348 (10; 98.47)	**0.0186 (9; 93.14)**	0.0399 (11; 98.98)
	True	0.0205 (14; 100.0)	0.0485 (9; 93.14)	**0.0049 (10; 98.47)**	0.0207 (15; 100.0)
Drupal	Infeasible	0.0944 (7; 89.56)	0.1231 (6; 86.69)	**0.0030 (21; 98.72)**	0.0070 (19; 98.31)
	Feasible	0.0744 (7; 89.56)	0.0175 (10; 95.26)	**0.0132 (18; 98.31)**	0.0296 (41; 99.95)
	True	0.0844 (7; 89.56)	0.0274 (10; 95.26)	**0.00500 (15; 98.49)**	0.0261 (10; 95.39)
SmartHome	Infeasible	0.0117 (17; 97,82)	0.0969 (7; 89,31)	**0.0040 (17; 98,60)**	0.3500 (3; 64,00)
	Feasible	0.0183 (17; 97,82)	**0.0015 (10; 96,06)**	0.0214 (18; 98,13)	0.0217 (18; 98,16)
	True	0.0044 (15; 97,55)	0.0194 (10; 96,06)	**0.0017 (17; 97,82)**	0.0852 (8; 89,48)

over NSGA-II) once these ones are similar to those ones found in each instance individually regarding to the percentage.

Regarding the runtime, Table 6 shows the mean time in seconds for all instances and algorithms. The bold values represent the smallest runtime.

We do not observe great runtime differences among the algorithms for the smallest instances. But R-NSGA-II presents the best performance (14 out of 18 cases), except for E-Shop, a case where NSGA-II is the best. But for the greatest instances Drupal and SmartHome, NSGA-II has a bad performance, it

Table 5: # of solutions for 2-objectives formulation.

FMRP		NSGA-II	RANDOM	r-NSGA-II	R-NSGA-II
James	Infeasible	1.00 (16.66%)	1.00 (100.00%)	1.36 (100.00%)	2.06 (96.38%)
	Feasible	3.00 (50.00%)	1.00 (100.00%)	1.03 (100.00%)	2.00 (94.44%)
	True	3.00 (50.00%)	1.00 (100.00%)	1.00 (100.00%)	2.00 (91.11%)
CAS	Infeasible	3.40 (34.29%)	1.66 (41.66%)	1.83 (100.00%)	7.80 (88.19%)
	Feasible	7.56 (70.60%)	1.66 (41.66%)	1.16 (100.00%)	7.20 (78.49%)
	True	6.90 (69.73%)	1.66 (41.66%)	1.20 (100.00%)	7.76 (86.02%)
WS	Infeasible	12.20 (84.55%)	2.63 (52.66%)	2.66 (100.00%)	9.33 (88.00%)
	Feasible	10.43 (72.18%)	2.00 (40.00%)	1.13 (100.00%)	8.76 (80.82%)
	True	10.43 (72.18%)	1.20 (24.00%)	1.26 (100.00%)	8.06 (80.38%)
E-Shop	Infeasible	12.10 (80.60%)	2.00 (22.19%)	3.93 (100.00%)	13.86 (92.84%)
	Feasible	10.86 (72.33%)	2.13 (23.27%)	1.70 (100.00%)	12.43 (86.06%)
	True	10.96 (73.01%)	2.66 (28.86%)	3.06 (100.00%)	12.60 (86.22%)
Drupal	Infeasible	16.03 (80.16%)	4.00 (38.86%)	4.96 (100.00%)	18.63 (90.82%)
	Feasible	15.46 (77.19%)	3.00 (29.15%)	4.53 (100.00%)	17.96 (88.73%)
	True	15.66 (78.20%)	3.00 (29.15%)	4.53 (100.00%)	17.50 (88.30%)
SmartHome	Infeasible	18.80 (82.89%)	4.10 (38.84%)	5.30 (100.00%)	20.66 (91.91%)
	Feasible	17.76 (78.45%)	3.03 (28.69%)	5.56 (100.00%)	19.46 (88.92%)
	True	18.03 (79.70%)	3.03 (28.69%)	5.70 (100.00%)	20.40 (90.55%)
Average	Infeasible	10.56 (63.18%)	2.55 (49.02%)	3.33 (100.00%)	12.05 (91.36%)
	Feasible	10.84 (70.11%)	2.13 (43.79%)	2.51 (100.00%)	11.29 (86.23%)
	True	10.82 (70.46%)	2.08 (42.05%)	2.79 (100.00%)	11.38 (87.09%)

takes more than the double of seconds than the PEMOAs. A possible reason to this is NSGA-II generates a greater number of solutions, outside the ROI and spends time with those uninteresting solutions.

Summarizing the results for 2-objectives formulation, it is possible to notice that, both PEMOAs are better options than NSGA-II considering all the indicators. The R-NSGA-II algorithm outperforms the other ones in most of instances and RPs considering R-HV. The R-NSGA-II solutions have more diversity and

Table 6: Runtime for 2-objectives formulation.

FMRP		NSGA-II	r-NSGA-II	R-NSGA-II
James	Infeasible	4.2048 (0.6593)	5.9306 (0.3399)	**3.8869 (0.3055)**
	Feasible	4.2048 (0.6593)	5.8439 (0.3445)	**3.8688 (0.3354)**
	True	4.2048 (0.6593)	5.8941 (0.3473)	**3.8021 (0.5573)**
CAS	Infeasible	6.1451 (0.5417)	7.5009 (0.3391)	**6.1222 (0.3032)**
	Feasible	**6.1451 (0.5417)**	7.5587 (0.3887)	6.1714 (0.4523)
	True	6.1451 (0.5417)	7.4334 (0.5660)	**5.9611 (0.3264)**
WS	Infeasible	6.4561 (0.4505)	8.7388 (0.4526)	**6.2261 (0.3720)**
	Feasible	6.4561 (0.4505)	8.1082 (2.2254)	**6.4266 (0.4338)**
	True	6.4561 (0.4505)	8.8411 (0.4285)	**6.3155 (0.4137)**
E-Shop	Infeasible	**10.8505 (0.5828)**	12.8305 (0.5052)	11.5651 (0.3151)
	Feasible	**10.8505 (0.5828)**	12.7548 (0.4654)	11.5349 (0.3378)
	True	**10.8505 (0.5828)**	13.0157 (0.3796)	11.2321 (0.8094)
Drupal	Infeasible	14111.5941 (3654.4270)	7229.6497 (126.0624)	**6764.4769 (105.3022)**
	Feasible	14111.5941 (3654.4270)	7209.0258 (149.7244)	**6790.2450 (108.8481)**
	True	14111.5941 (3654.4270)	7148.4222 (105.4341)	**6689.5100 (367.3866)**
SmartHome	Infeasible	23443.5364 (2859.9571)	9596.4837 (156.5624)	**9004.1140 (150.9896)**
	Feasible	23443.5364 (2859.9571)	9574.2591 (234.9503)	**9154.3924 (223.5698)**
	True	23443.5364 (2859.9571)	9487.9919 (123.7744)	**8645.7928 (102.1078)**
Average	Infeasible	15666.9178 (1086.1031)	2810.1890 (47.3769)	2632.7318 (42.9312)
	Feasible	15666.9178 (1086.1031)	2802.9251 (64.6831)	2662.1065 (55.6629)
	True	15666.9178 (1086.1031)	2778.5997 (38.4883)	2560.4356 (78.6002)

convergence when compared with r-NSGA-II. Considering the number of solutions in the ROI, R-NSGA-II presents the best trade-off between the number
670 of interesting and uninteresting solutions. Besides, if the DM is interested to visualize few solutions r-NSGA-II seems to be a good option because in almost all cases 100% of the r-NSGA-II solutions are in the ROI. Regarding to the runtime, R-NSGA-II should be considered once it can generate the solutions faster even when the number of products to be selected increases.

6.2. 3-Objectives Formulation

Table 7 shows the mean values and standard deviations for R-HV indicator. Again, the bold values represent the best ones, and light gray cells represent values that are statistically equivalent.

Table 7: R-HV for 3-objectives formulation.

FMRP		**NSGA-II**	**RANDOM**	**r-NSGA-II**	**R-NSGA-II**
James	Infeasible	0.3848 (0.1994)	0.0350 (0.0000)	0.5257 (0.0924)	**0.8133 (0.0070)**
	Feasible	0.7358 (0.1624)	0.0378 (0.0000)	0.8844 (0.0331)	**0.8906 (0.0180)**
	True	0.7587 (0.1579)	0.0361 (0.0000)	0.7893 (0.0309)	**0.8609 (0.0073)**
CAS	Infeasible	0.9442 (0.0104)	0.4497 (0.0332)	0.8416 (0.0604)	**0.9502 (0.0086)**
	Feasible	0.9267 (0.0236)	0.4528 (0.0336)	0.8870 (0.0314)	**0.9406 (0.0176)**
	True	0.9577 (0.0110)	0.4479 (0.0348)	0.9130 (0.0422)	**0.9651 (0.0066)**
WS	Infeasible	0.8879 (0.0242)	0.49515 (0.0330)	**0.9561 (0.0073)**	0.9473 (0.0151)
	Feasible	0.9673 (0.0074)	0.4850 (0.0319)	0.9464 (0.0066)	**0.9728 (0.0060)**
	True	0.9576 (0.0078)	0.4901 (0.0314)	0.9491 (0.0108)	**0.9728 (0.0056)**
E-Shop	Infeasible	0.9660 (0.0047)	0.7628 (0.0280)	0.9639 (0.0066)	**0.9763 (0.0070)**
	Feasible	**0.9808 (0.0062)**	0.7645 (0.0281)	0.9507 (0.0127)	0.9785 (0.0074)
	True	0.9583 (0.0075)	0.7831 (0.0292)	0.9517 (0.0088)	**0.9800 (0.0058)**
Drupal	Infeasible	0.9853 (0.0038)	0.8311 (0.0271)	0.9922 (0.0033)	**0.9943 (0.0030)**
	Feasible	0.9717 (0.0091)	0.8480 (0.0294)	0.9540 (0.0084)	**0.9851 (0.0093)**
	True	0.9924 (0.0028)	0.8466 (0.0278)	0.9786 (0.0041)	**0.9938 (0.0032)**
SmartHome	Infeasible	0.9811 (0.0036)	0.83518 (0.0223)	0.9828 (0.00346)	**0.9914 (0.0030)**
	Feasible	0.9721 (0.0111)	0.87129 (0.0225)	0.9500 (0.01138)	**0.9859 (0.0086)**
	True	0.9914 (0.0031)	0.83447 (0.0223)	0.9811 (0.00375)	**0.9918 (0.0030)**

The results of R-HV are similar to the ones obtained with 2-objectives. R-NSGA-II generates the best results in most instances and RPs (16 out of 18 cases). In only one case (instance E-Shop), the traditional NSGA-II generates the best result (but with no statistical difference with R-NSGA-II). Regarding r-NSGA-II, it generates the best results in a case for WS. When the number of products to be selected increases, the R-NSGA-II remains to be the best option with the best results. Regarding to random algorithm, it had the worst performance in all cases.

Table 8 shows the ED values and the selected solution (the closest one to the RPs) for 3-objectives formulation. The bold values represent the best ones.

Table 8: ED for 3-objectives formulation.

FMRP		**NSGA-II**	**RANDOM**	**r-NSGA-II**	**R-NSGA-II**
James	Infeasible	0.2273 (3; 75.47; 98.66)	0.7039 (1; 32.07; 73.33)	0.2273 (3; 75.47; 98.66)	**0.0810 (4; 91.50; 100.0)**
	Feasible	**0.0283 (6; 100.0; 100.0)**	0.7077 (1; 32.07; 73.33)	0.0631 (4; 92.45; 98.66)	0.2029 (3; 78.30; 96.00)
	True	**0.0524 (6; 100.0; 100.0)**	0.7045 (1; 32.07; 73.33)	0.0608 (4; 92.45; 100.0)	0.2065 (3; 77.35; 97.33)
CAS	Infeasible	0.2796 (3; 70.04; 97.81)	0.2224 (4; 75.77; 97.26)	**0.1344 (4; 84.58; 97.26)**	0.2447 (3; 73.56; 96.72)
	Feasible	0.0735 (5; 89.86; 100.0)	0.2029 (4; 75.77; 97.26)	**0.0423 (7; 97.35; 100.0)**	0.1196 (4; 84.14; 97.26)
	True	0.0819 (5; 89.86; 100.0)	0.2230 (4; 75.77; 97.26)	**0.0289 (6; 95.15; 99.45)**	0.1917 (4; 78.85; 100.0)
WS	Infeasible	0.2434 (4; 74.51; 87.69)	0.3898 (3; 65.26; 74.35)	0.0997 (6; 88.51; 91.79)	**0.0322 (8; 94.39; 95.38)**
	Feasible	0.0286 (14; 100.0; 100.0)	0.1903 (5; 80.67; 90.25)	**0.0175 (11; 99.44; 98.97)**	0.0330 (9; 96.07; 95.38)
	True	0.0253 (14; 100.0; 100.0)	0.1942 (5; 80.67; 90.25)	**0.0066 (10; 97.47; 98.97)**	0.1833 (5; 83.19; 88.20)
E-Shop	Infeasible	0.0789 (7; 91.11; 99.01)	0.1293 (8; 87.05; 94.05)	**0.0238 (9; 97.20; 97.52)**	0.0281 (10; 96.44; 100.0)
	Feasible	0.0697 (7; 91.11; 99.01)	0.1164 (8; 87.05; 94.05)	**0.0160 (11; 99.23; 99.01)**	0.0423 (7; 93.90; 97.03)
	True	0.0597 (7; 91.11; 99.01)	0.1070 (8; 87.05; 94.05)	**0.0366 (8; 93.40; 98.51)**	0.1721 (4; 81.47; 90.59)
Drupal	Infeasible	0.0120 (26; 98.13; 99.81)	0.0386 (11; 95.67; 97.04)	**0.0055 (24; 98.99; 99.53)**	0.0317 (15; 95.94; 98.15)
	Feasible	0.0303 (26; 98.13; 99.81)	**0.0134 (11; 95.67; 97.04)**	0.0198 (21; 98.26; 98.52)	0.0929 (8; 88.46; 93.34)
	True	0.0083 (26; 98.13; 99.81)	0.0305 (11; 95.67; 97.04)	**0.0052 (18; 97.49; 98.89)**	0.0271 (13; 96.03; 97.13)
SmartHome	Infeasible	0.1979 (6; 81.85; 89.12)	0.0300 (10; 96.06; 98.42)	0.0096 (17; 98.13; 98.59)	**0.0082 (21; 98.50; 99.64)**
	Feasible	0.0480 (37; 99.79; 99.94)	0.0143 (10; 96.06; 98.42)	**0.0252 (16; 97.52; 99.00)**	0.0300 (21; 98.37; 98.83)
	True	0.0125 (37; 99.79; 99.94)	0.0299 (10; 96.06; 98.42)	**0.0042 (22; 99.05; 99.41)**	0.1352 (7; 86.09; 94.97)

The table shows that NSGA-II outperforms the other algorithms just for James, with feasible and true RPs. The random algorithm is the best for feasible RP in Drupal and R-NSGA-II for infeasible RPs in James, WS and SmartHome instances. For the other instances, we can see that r-NSGA-II generates the best results, highlighting this one reached the best performance in most instances (12 out of 18). These results are similar to those ones with 2-objectives.

Table 9: # of solutions for 3-objectives formulation.

FMRP		**NSGA-II**	**RANDOM**	**r-NSGA-II**	**R-NSGA-II**
James	Infeasible	3.00 (28.37%)	1.00 (100.00%)	2.80 (97.50%)	3.20 (72.73%)
	Feasible	3.60 (33.78%)	1.00 (100.00%)	1.53 (100.00%)	2.96 (75.16%)
	True	3.60 (33.78%)	1.00 (100.00%)	1.56 (86.33%)	3.53 (80.60%)
CAS	Infeasible	14.66 (43.74%)	2.93 (43.95%)	3.93 (100.00%)	13.36 (70.38%)
	Feasible	10.50 (31.20%)	3.00 (45.06%)	1.23 (100.00%)	10.23 (65.07%)
	True	14.00 (41.81%)	2.93 (43.95%)	2.53 (100.00%)	11.96 (60.89%)
WS	Infeasible	25.73 (59.40%)	3.93 (46.84%)	7.80 (100.00%)	17.90 (80.55%)
	Feasible	20.73 (47.88%)	3.73 (44.38%)	1.93 (100.00%)	14.83 (73.89%)
	True	25.13 (58.07%)	3.73 (44.38%)	6.40 (100.00%)	16.70 (71.29%)
E-Shop	Infeasible	20.53 (67.38%)	8.23 (54.84%)	6.43 (100.00%)	20.03 (82.07%)
	Feasible	19.06 (62.57%)	6.60 (44.11%)	2.36 (100.00%)	21.26 (80.28%)
	True	22.63 (74.33%)	8.66 (57.60%)	5.86 (100.00%)	19.46 (90.62%)
Drupal	Infeasible	22.90 (80.18%)	3.80 (23.68%)	6.46 (100.00%)	24.00 (88.21%)
	Feasible	22.00 (76.93%)	3.60 (22.45%)	4.66 (100.00%)	25.10 (88.14%)
	True	22.26 (77.96%)	3.60 (22.45%)	5.66 (100.00%)	25.16 (88.66%)
SmartHome	Infeasible	25.80 (83.03%)	5.76 (34.54%)	8.40 (100.00%)	28.83 (90.81%)
	Feasible	24.53 (78.90%)	4.40 (26.10%)	5.16 (100.00%)	27.56 (88.95%)
	True	24.76 (79.61%)	4.43 (26.27%)	6.20 (100.00%)	28.33 (90.21%)
Average	Infeasible	19.26 (60.34%)	4.26 (50.63%)	5.96 (99.55%)	17.88 (80.78%)
	Feasible	16.73 (55.20%)	3.71 (47.00%)	2.81 (100.00%)	16.98 (78.57%)
	True	18.72 (75.87%)	4.05 (49.10%)	4.69 (97.71%)	17.52 (80.37%)

695 Regarding the number of found solutions, see Table 9. Similarly to what happens for the 2-objectives formulation, the table shows the random algorithm can generate the smallest number of solutions in most instances (12 out of 18). However, most of them are outside the ROI (in some cases, only 22% of them are inside the ROI). The r-NSGA-II algorithm generates the smallest number

of solutions and all of them (100%) are inside the ROI. There is only one case, instance James with infeasible RP, that this percentage is 97%.

Again, it is possible to see that NSGA-II has a similar performance comparing to R-NSGA-II regarding the number of solutions in the ROI. However, it seems NSGA-II takes time to generate solutions that are uninteresting for the DM, for example, when the RP is feasible, it generates, on average, just 55% of solutions inside the ROI. At the end, both algorithms generate almost the same number of solutions but R-NSGA-II generates most of them inside the ROI (a percentage around 80% considering all RPs). Again, we do not observe a difference in the algorithms' performance regarding the number of products.

Regarding runtime, Table 10 shows the mean time in seconds for all instances and algorithms. Again, the bold values represent the smallest runtime.

R-NSGA-II takes the smaller runtime for all RPs and instances, except for E-Shop where NSGA-II performed better. However, for E-Shop instance, NSGA-II outperforms the others. Similar result was found with 2-objectives. This can suggest that when the number of products to be selected increases, PEMOAs should be considered, because the traditional algorithms spends effort with uninteresting solutions outside the ROI.

Summarizing the results for 3-objectives formulation, it is possible to notice that the performance remains similar when compared to 2-objectives formulation. The PEMOAs are better than the traditional NSGA-II. The R-NSGA-II algorithm outperforms the other ones in most instances and RPs regarding R-HV, that is, regarding diversity and convergence. On the other hand, r-NSGA-II presented better results considering the indicator ED. With respect to the number of solutions in the ROI, again, r-NSGA-II generates the smallest numbers, with all solutions inside the ROI. R-NSGA-II presents the best trade-offs, and should be used if the DM wants to visualize more solutions inside the ROI. However, if the DM is interested to select just one solution close to provided RP, r-NSGA-II should be considered. Regarding to the runtime, the results are similar to 2-objectives, that is, R-NSGA-II should be considered once it can generate the solutions faster even when the number of products to be selected

Table 10: Runtime for 3-objectives formulation.

FMRP		NSGA-II	r-NSGA-II	R-NSGA-II
James	Infeasible	4.5149 (0.7025)	6.7518 (0.3913)	**4.1322 (0.3364)**
	Feasible	4.5149 (0.7025)	6.6088 (0.3227)	**4.2709 (0.2993)**
	True	4.5149 (0.7025)	6.7665 (0.4067)	**3.9837 (0.7157)**
CAS	Infeasible	6.8239 (0.4941)	8.9293 (1.7383)	**6.7269 (0.4512)**
	Feasible	6.8239 (0.4941)	7.4641 (3.8303)	**6.6388 (0.4699)**
	True	6.8239 (0.4941)	8.8919 (0.3860)	**6.6355 (0.3847)**
WS	Infeasible	7.7705 (0.4791)	10.4503 (0.4338)	**7.3108 (0.3970)**
	Feasible	7.7705 (0.4791)	10.6640 (0.5586)	**7.3778 (0.4135)**
	True	7.7705 (0.4791)	10.5872 (0.4118)	**7.5032 (0.4421)**
E-Shop	Infeasible	**13.2289 (0.6239)**	16.7447 (0.5031)	14.1021 (0.4817)
	Feasible	**13.2289 (0.6239)**	16.7650 (0.4023)	14.0769 (0.3808)
	True	**13.2289 (0.6239)**	16.6181 (0.4453)	13.8140 (1.3895)
Drupal	Infeasible	13792.1982 (279.1871)	10665.3864 (130.1581)	**9962.0670 (147.6519)**
	Feasible	13792.1982 (279.1871)	10674.9614 (170.3901)	**9923.7371 (184.4256)**
	True	13792.1982 (279.1871)	10615.4942 (131.8898)	**9450.8956 (1236.2752)**
SmartHome	Infeasible	25334.0734 (5538.5321)	15174.3424 (219.6652)	**14331.6325 (223.0234)**
	Feasible	25334.0734 (5538.5321)	14976.6581 (213.6948)	**14391.8361 (200.4549)**
	True	25334.0734 (5538.5321)	15275.3411 (165.1807)	**14097.5451 (460.1420)**
Average	Infeasible	6526.4350(970.0031)	4313.7675 (58.8150)	4054.3286 (62.0569)
	Feasible	6526.4350(970.0031)	4282.1869 (64.8665)	4057.9896 (64.4073)
	True	6526.4350(970.0031)	4322.2831 (49.7867)	3930.0628 (283.2248)

increases.

6.3. Answering RQ1 (PEMOAs × NSGA-II × Random)

The random selection was used as a baseline and presented the worst results in almost all cases and indicators, showing the feasibility and importance of our approach.

In the great majority of the cases, the PEMOAs (r-NSGA-II and R-NSGA-II) perform better than the traditional NSGA-II, considering all the indicators.

This is possible because these algorithms have a search mechanism focused on finding good solutions towards the RP, while NSGA-II generates solutions that can be considered not so interesting from the DM's point of view. As a consequence, PEMOAs can generate a greater percentage of solutions inside the ROI for both formulations, independently of the kind of RP provided and the size of the FM instances.

Regarding runtime, the performance was similar for both formulations, that is, PEMOAs executed faster than NSGA-II algorithm, mainly for the larger instances. In such cases the solution space is greater and it seems that NSGA-II spends time with uninteresting solutions. Then the results suggest the PEMOAs should be considered when the number of products to be selected increases.

To illustrate this advantage, we take the results for WS with infeasible RP, obtained by executing each algorithm only once with the same parameters adopted in our experiment (2-objectives formulation). The frontiers corresponding to the sets PF_{known} of each algorithm and the set PF_{true} are presented in Figure 6. Analyzing the figure it is possible to notice and confirm that the NSGA-II algorithm can generate more solutions in the true Pareto front. However, some of them are very far from the RP while PEMOAs generate solutions closest to RPs.

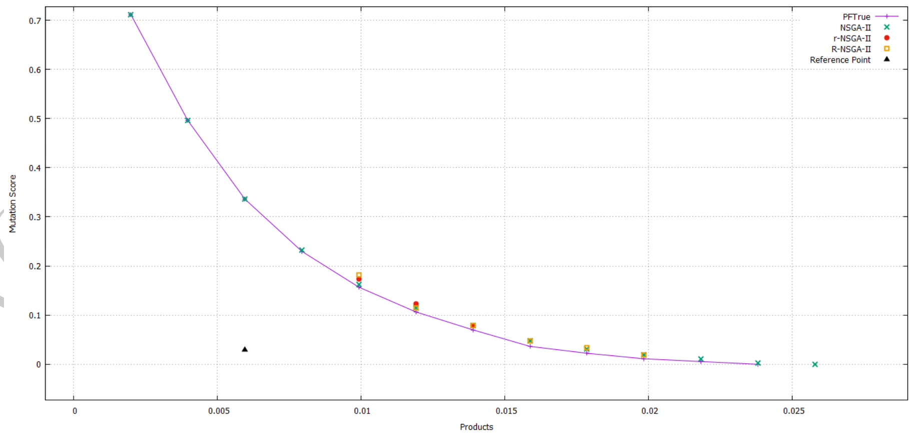

Figure 6: Example of solutions for the WS instance using the infeasible RP.

Continuing the analysis Table 11 presents the number of solutions generated by each algorithm inside the ROI and the number outside the ROI. NSGA-II generates 12 possible solutions but only 4 ones (33%) are inside the ROI, almost 67% of them can be uninteresting from the tester's point of view. On the other hand, R-NSGA-II generates 4 (66%) solutions inside the ROI, while r-NSGA-II generates 3 possible solutions (100%) inside the ROI.

Table 11: Number of solutions by algorithm for WS using an infeasible RP

Algorithm	**Inside ROI**	**Uninteresting**
NSGA-II	4	8
r-NSGA-II	3	0
R-NSGA-II	4	2

This finding can also be observed for all the other instances, independently of the RPs and number of used objectives. Taking now the frontiers obtained to the largest instance SmartHome with true RP (Figure 7) it is possible to see that the NSGA-II solutions are distributed along the true Pareto front and some of them are very far from the RP.

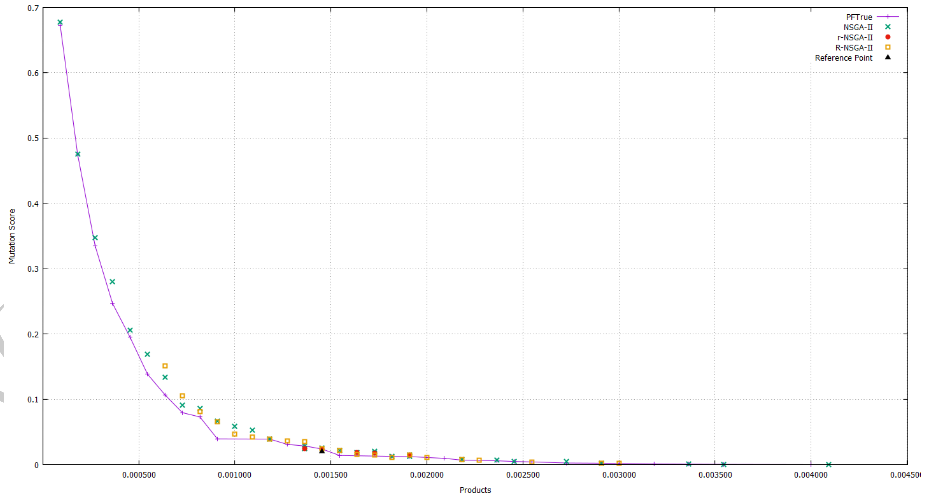

Figure 7: Example of solutions for the SmartHome instance for true RP.

Therefore, it is more convenient and easy for the DM to select the best solutions by using the ones generated by the PEMOAs. According to Table 12. NSGA-II generates 28 possible solutions, 18 (64%) are inside the ROI, almost 36% of them can be uninteresting from the tester's point of view. Regarding to R-NSGA-II, this one generates 21 (85%) solutions inside the ROI (and 15% outside), while r-NSGA-II generates 5 possible solutions (100%) inside the ROI. In this sense, the use of the PEMOAs is advantageous. It is clear that besides the time spent by the traditional algorithm outside the ROI, it requires more effort from the tester, who needs to analyse a greater number of solutions with low coverage, which are, in most cases, uninteresting. The greater the instance the greater this number tends to be.

Table 12: Number of solutions by algorithm for Smarthome using an true RP

Algorithm	**Inside ROI**	**Uninteresting**
NSGA-II	18	10
r-NSGA-II	5	0
R-NSGA-II	18	3

6.4. Answering RQ2

The R-NSGA-II algorithm can be considered as the best option if the DM is interested to select or visualize more solutions near to the RP, i.e., it can generate more solutions inside the ROI. These findings are easily demonstrated when we take into account the values for R-HV and the percentage of solutions inside the ROI. We can observe that the performance of r-NSGA-II and R-NSGA-II remains almost the same when the number of objectives or number of products increases.

However, r-NSGA-II obtains the best ED values and if the DM is interested in the selection of just a few solutions, such an algorithm should be considered because it generates the smallest number of solutions in the ROI and, in most cases, it does not generate any solution outside. This behavior happens for both formulations.

Regarding to the runtime, r-NSGA-II is a little bit slower comparing to R-NSGA-II for all instances and RPs. However, when the number of products to be selected increases, this difference tends to decrease.

6.5. Discussion

The implemented mechanism to use a set of n products makes our approach scalable for large FMs. However, two questions arise regarding such set: i) how to choose n, and ii) how to select the initial products.

In this paper, they were chosen considering the required score (97%) and the products were randomly selected. Such strategy is easy to be implemented and adopted by the testers. It reached good results in our experiments with Drupal and SmartHome, but other strategies can be adopted, according to the tester's needs and development environment. For example, some products that cover the most relevant features can be included in the initial population, some non-functional properties associated to the FM can also be considered, PEMOAs may be used to prioritize a subset of the products of the FM, and so on.

Such elements should be provided a priori as part of the RP-based approach, or we can also use them in the fitness function. We intend to evaluate other strategies in future works.

7. Threats to Validity

In our approach, we used the tool FMTS, which makes use of the FaMa framework to deal with resource model constraints and derive products. However, the approach is independent of the solver being used and could be used without any modifications with SAT solvers, for example. Other objectives could be used as well. To do this it is only necessary to implement other procedures to calculate the input matrices.

In our evaluation, most of the FMs are small, when compared to the ones used in the industry. Scalability can be a limitation. To minimize such a threat we implemented a mechanism to work with a reduced number of products.

This mechanism was evaluated with two large FMs and we obtained promising results. But it was executed only once and the selected products were used for 30 executions and this can be considered as a threat. The obtained mean execution time is, in the worst case, around 2.7 hours. This time is acceptable to obtain solutions that satisfy the user preferences. In such cases, a number n of 11000 products were used for the large SPLs. We consider that this number was adequate proportionally to the number of products and the mutation score (around 97%) chosen for the experiments performed in this paper. However, a greater number for even large SPLs should be evaluated in future experiments. We expect that the approach continues obtaining good results with acceptable runtimes.

The RPs were selected considering our previous knowledge about the Pareto-front generated by NSGA-II. They do not represent real RPs determined by the tester. To better evaluate the impact of the provided RPs, the approach should be evaluated with real SPLs in the industrial environment. To minimize such a threat we analyzed three kinds of RPs, simulating possible solutions desired by the tester, who, in practice, has no previous knowledge about the fronts. The approach presented a good performance for all RPs. But maybe different results can be obtained with other RPs. The use of more than one RP in each algorithm execution can impact (or even improve) the results.

In our work, we generated the initial population at random. We randomly chose the number of products in each individual considering a percentage of the number of products being considered n, but other ways to generate the population can be adopted. It is possible to use seeds that can improve the performance of the algorithms, mainly when large FMs are being tested. This could lead to reduced runtime and should be evaluated in future experiments.

Another limitation is that for the larger FMs some mutants that could not be killed by the generated products were considered equivalent. Maybe some of them could be killed if all products are used. Equivalent mutant detection for programs is indeed undecidable. However, in the SPL context, this is decidable but not scalable. The percentage of equivalent mutants may depend on the

FM characteristics more than the number of products [49]. To minimize such a threat we generated products to ensure the discarding of around 3% of mutants, considering the percentage found for the smaller FMs determined using the complete set of products.

Finally, the performance of the algorithms can depend on the parameters. Other parameter settings can generate different results. To minimize such a threat, we performed a tuning phase. Due to the stochastic nature of the algorithms, we run each algorithm 30 times.

8. Concluding Remarks

This work introduces a PEMOA approach to derive products for the SPL testing. The approach is multi-objective and allows the use of different factors that impact the problem, taking into account the preferences and needs of the tester. As another advantage, the approach is a priori and based on the RP method. This makes it different from other approaches found in the SBSE field, which are mono-objective, and interactive, requiring the DM's intervention many times during the optimization process, which can cause fatigue.

We observe that the RP method is very suitable to our problem because many times a required coverage level or cost budget exists. Knowing that, the algorithms must have to avoid the generation of a lot of uninteresting solutions from the tester's point of view, reducing efforts.

To evaluate our approach we use two PEMOAs, r-NSGA-II and R-NSGA-II, which are based on the traditional NSGA-II, largely employed in the SBSE field, and that reach good results reported by works addressing the same problem. Our analysis takes into account different kind of RPs, and sets of two and three objectives to reach a minimum possible number of products with high pairwise coverage and mutation score.

The PEMOAs outperformed the traditional NSGA-II by generating a smaller number of solutions and most of them inside the ROI, i.e. near to the RPs. Besides, R-NSGA-II presented the best performance considering the R-HV indicator, and produced the greatest number of solutions in the ROI. On the other hand, if the DM is interested to visualize few solutions inside the ROI, the r-NSGA-II algorithm should be considered. We did not observe great differences when using different sets of objectives and RPs. Also, the performance of the PEMOAs tends to remains the same when the number of objectives increases.

Future experiments should be also conducted with a set of larger SPLs and considering real cases in the industry. In such scenarios, our approach could be also compared with an interactive one. To this end, it is necessary to investigate the best way to incorporate the preferences interactively, that is, by providing a ranking number or a set of features, by choosing a specific solution, by providing a bound for a total number of products for large FMs, by providing some pairs, considered relevant for a given FM, and so on.

Our approach could be used with other sets of objectives and PEMOAs. For instance, there are other preference algorithms based on NSGA-II that could be used such as p-NSGA-II [59] and g-NSGA-II [60]. Other research opportunities are related to the adaptation of other EMOAs with concepts explored here. The SPEA-II algorithm can be adapted to use the r-dominance concept, as well as the algorithm IBEA to use the R-HV indicator. Besides, in the former context, some indicator-based algorithms such as SATIBEA [61] and SMTIBEA [62] could be evaluated once these ones reached good results and they could be used as a simple way to provide a set of configurations at the initialization of the algorithm.

In addition to this, the approach could be explored to solve similar software testing problems or other ones from the software engineering area. The preliminary results herein presented can motivate other works and contribute to the PSBSE field.

9. Acknowledgments

This work is supported by Brazilian agencies CAPES and CNPq, grants: 307762/2015-7 and 473899/2013-2.

References

[1] K. C. Kang, J. Lee, P. Donohoe, Feature-oriented project line engineering, IEEE Software 19 (4) (2002) 58–65.

[2] M. B. Cohen, M. B. Dwyer, J. Shi, Coverage and adequacy in software product line testing, in: Proceedings of the Workshop on Role of Software Architecture for Testing and Analysis (ROSATEA'06), ACM, Portland, USA, 2006, pp. 53–63.

[3] S. Rapps, E. J. Weyuker, Selecting software test data using data flow information, IEEE Transactions on Software Engineering 11 (4) (1985) 367–375.

[4] B. P. Lamancha, M. P. Usaola, Testing product generation in software product lines using pairwise for features coverage, in: Proceedings of the 22nd International Conference on Testing Software and Systems (ICTSS'10), Springer, Natal, Brazil, 2010, pp. 111–125.

[5] 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 International Conference on Software Testing, Verification and Validation (ICST'10), IEEE, Paris, France, 2010, pp. 459–468.

[6] 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 Transactions on Software Engineering 40 (7) (2014) 650–670.

[7] J. McGregor, Testing a software product line, Tech. Rep. CMU/SEI-2001-TR-022, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (2001).

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

[9] 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), Vol. 2, ACM, Munich, Germany, 2011, pp. 1–8.

[10] E. Uzuncaova, S. Khurshid, D. Batory, Incremental test generation for software product lines, IEEE Transactions on Software Engineering 36 (3) (2010) 309–322.

[11] 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.

[12] 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 International Conference on Software Testing, Verification and Validation (ICSTW'13), IEEE, Luxembourg, Luxembourg, 2013, pp. 188–197.

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

[14] H. Lackner, M. Schmidt, Towards the assessment of software product line tests: A mutation system for variable systems, in: Proceedings of the 18th International Software Product Line Conference (SPLC'14), Vol. 2, 2014, pp. 62–69.

[15] 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), Nashville, USA, 2015, pp. 131–140.

[16] M. Harman, S. A. Mansouri, Y. Zhang, Search-based software engineering: Trends, techniques and applications, Computing Surveys 45 (1) (2012) 1–61.

[17] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, Y. Zhang, Search based software engineering for software product line engineering: A survey and directions for future work, in: Proceedings of the 18th International Software Product Line Conference (SPLC'14), Vol. 1, ACM, Florence, Italy, 2014, pp. 5–18.

[18] R. E. Lopez-Herrejon, L. Linsbauer, A. Egyed, A systematic mapping study of search-based software engineering for software product lines, Information and Software Technology 61 (C) (2015) 33–51.

[19] 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: International Conference on Software Maintenance (ICSM'13), 2013, pp. 404–407.

[20] 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.

[21] R. A. Matnei Filho, S. R. Vergilio, A multi-objective test data generation approach for mutation testing of feature models, Journal of Software Engineering Research and Development 4 (1) (2016) 1–29.

[22] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.

[23] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization, in: Evolutionary

Methods for Design Optimization and Control with Applications to Industrial Problems, International Center for Numerical Methods in Engineering, Athens, Greece, 2001, pp. 95–100.

[24] E. Zitzler, S. Künzli, Indicator-based selection in multiobjective search, in: Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII), Springer, Birmingham, UK, 2004, pp. 832–842.

[25] T. N. Ferreira, S. R. Vergilio, J. T. de Souza, Incorporating user preferences in search-based software engineering: A systematic mapping study, Information and Software Technology 90 (Supplement C) (2017) 55–69.

[26] H. Takagi, Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation, Proceedings of the IEEE 89 (9) (2001) 1275–1296.

[27] J. Branke, K. Deb, K. Miettinen, R. Słowinski, Multiobjective Optimization - Interactive and Evolutionary Approaches, Vol. 5252, Springer-Verlag, New York, 2008.

[28] K. Deb, J. Sundar, U. Bhaskara, S. Chaudhuri, Reference point based multi-objective optimization using evolutionary algorithms, International Journal of Computational Intelligence Research 2 (3) (2006) 27–286.

[29] L. B. Said, S. Bechikh, K. Ghédira, The r-dominance: A new dominance relation for interactive evolutionary multicriteria decision making, IEEE Transactions on Evolutionary Computation 14 (5) (2010) 801–818.

[30] J. Branke, K. Deb, K. Miettinen, Multiobjective optimization: Interactive and evolutionary approaches, Vol. 5252, Springer Science & Business Media, 2008.

[31] A. P. Wierzbicki, The use of reference objectives in multiobjective optimization, in: Multiple criteria decision making theory and application, Springer, 1980, pp. 468–486.

[32] S. Bechikh, M. Kessentini, L. B. Said, K. Ghédira, Preference incorporation in evolutionary multiobjective optimization: A survey of the state-of-the-art, in: A. R. Hurson (Ed.), Advances in Computers, Vol. 98, Elsevier, 2015, pp. 141 – 207.

[33] K. Li, K. Deb, X. Yao, R-metric: Evaluating the performance of preference-based evolutionary multi-objective optimization using reference points, IEEE Transactions on Evolutionary Computation (2017) 1–15 Early access, DOI: 10.1109/TEVC.2017.2737781.

[34] 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, Journal of System and Software 122 (2016) 287–310.

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

[36] 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), Vol. 1, 2014, pp. 32–41.

[37] F. Ensan, E. Bagheri, D. Gašević, Evolutionary search-based test generation for software product line feature models, in: Proceedings of the 24th International Conference on Advanced Information Systems Engineering (CAiSE'12), Springer, Gdansk, Poland, 2012, pp. 613–628.

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

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

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

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

[42] T. N. Ferreira, J. A. Prado Lima, A. Strickler, J. N. Kuk, S. R. Vergilio, A. Pozo, Hyper-heuristic based product selection for software product line testing, IEEE Computational Intelligence Magazine 12 (2) (2017) 34–45.

[43] B. Marculescu, R. Feldt, R. Torkar, S. Poulding, An initial industrial evaluation of interactive search-based testing for embedded software, Applied Soft Computing 29 (2015) 26–39.

[44] R. Feldt, An interactive software development workbench based on biomimetic algorithms, Tech. Rep. abs/1211.5451, Chalmers University of Technology, Gothenburg, Sweden (November 2002).

[45] S. Kalboussi, S. Bechikh, M. Kessentini, L. Ben Said, Preference-based many-objective evolutionary testing generates harder test cases for autonomous agents, in: Proceedings of the 5th International Symposium on Search Based Software Engineering (SSBSE'13), Springer-Verlag New York, Inc., New York, NY, USA, 2013, pp. 245–250.

[46] A. E. E. Yamany, M. Shaheen, A. S. Sayyad, OPTI-SELECT: An interactive tool for user-in-the-loop feature selection in software product lines, in:

Proceedings of the 18th International Software Product Line (SPLC'14), Florence, Italy, 2014, pp. 126–129.

[47] A. E. E. Yamany, M. S. Elgamel, Smart Optiselect preference based innovative framework for user-in-the-loop feature selection in software product lines, in: Proceedings of the 7th International Conference on Information Technology (ICIT'15), 2015, pp. 657–666.

[48] 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, pp. 359–359.

[49] J. M. Ferreira, S. R. Vergilio, M. A. Quináia, A mutation approach to feature testing of software product lines, International Journal of Software Engineering and Knowledge Engineering (IJSEKE) (2017) 232–237.

[50] J. J. Durillo, A. J. Nebro, jmetal: A java framework for multi-objective optimization, Advances in Engineering Software 42 (10) (2011) 760–771.

[51] E. Zitzler, Evolutionary algorithms for multiobjective optimization: Methods and applications, Ph.D. thesis, ETH Zurich, Switzerland (1999).

[52] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, V. G. da Fonseca, Performance assessment of multiobjective optimizers: An analysis and review, IEEE Transactions on Evolutionary Computation 7 (2) (2003) 117–132.

[53] W. H. Kruskal, W. A. Wallis, Use of ranks in one-criterion variance analysis, Journal of the American Statistical Association 47 (260) (1952) 583–621.

[54] M. Mendonça, M. Branco, D. Cowan, SPLOT: software product lines online tools, in: 24th ACM SIGPLAN Conference Companion on Object oriented programming systems languages and applications, 2009, pp. 761–762.

[55] D. Benavides, S. Trujillo, P. Trinidad, On the modularization of feature models, in: First European Workshop on Model Transformation, 2005, p. 134.

[56] S. Weißleder, D. Sokenou, H. 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), 2008, p. 19.

[57] D. Beuche, M. Dalgarno, Software product line engineering with feature models, Overload Journal 78 (2012) 5–8, http://www.pure-systems.com/fileadmin/downloads/pure-variants/tutorials/SPLWithFeatureModelling.pdf.

[58] S. Segura, R. M. Hierons, D. Benavides, A. Ruiz-Cortés, Automated test data generation on the analyses of feature models: A metamorphic testing approach, in: Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST'10), 2010, pp. 35–44.

[59] J. Hu, G. Yu, J. Zheng, J. Zou, A preference-based multi-objective evolutionary algorithm using preference selection radius, Soft Computing 21 (17) (2017) 5025–5051.

[60] J. Molina, L. V. Santana, A. G. Hernández-Díaz, C. A. C. Coello, R. Caballero, g-dominance: Reference point based dominance for multiobjective metaheuristics, European Journal of Operational Research 197 (2) (2009) 685 – 692.

[61] C. Henard, M. Papadakis, M. Harman, Y. Le Traon, Combining multi-objective search and constraint solving for configuring large software product lines, in: International Conference on Software Engineering (ICSE), Vol. 1, 2015, pp. 517–528.

[62] J. Guo, J. H. Liang, K. Shi, D. Yang, J. Zhang, K. Czarnecki, V. Ganesh, H. Yu, SMTIBEA: a hybrid multi-objective optimization algorithm for configuring large constrained software product lines, Software & Systems Modeling (2017) 1–20.