

Article

Software Product Line Maintenance Using Multi-Objective Optimization Techniques

Muhammad Abid Jamil ^{1,*}, Mohamed K. Nour ¹, Saud S. Alotaibi ², Mohammad Javed Hussain ²,
Syed Mutiullah Hussaini ¹ and Atif Naseer ³

¹ Department of Computer Science, College of Computers and Information Systems, Umm Al Qura University, Makkah 21955, Saudi Arabia; mknour@uqu.edu.sa (M.K.N.); sshussaini@uqu.edu.sa (S.M.H.)

² Department of Information Systems, College of Computers and Information Systems, Umm Al Qura University, Makkah 21955, Saudi Arabia; ssotaibi@uqu.edu.sa (S.S.A.); mjhussain@uqu.edu.sa (M.J.H.)

³ Science and Technology Unit, Umm Al Qura University, Makkah 21955, Saudi Arabia; anahmed@uqu.edu.sa

* Correspondence: majamil@uqu.edu.sa

Abstract: Currently, software development is more associated with families of configurable software than the single implementation of a product. Due to the numerous possible combinations in a software product line, testing these families of software product lines (SPLs) is a difficult undertaking. Moreover, the presence of optional features makes the testing of SPLs impractical. Several features are presented in SPLs, but due to the environment's time and financial constraints, these features are rendered unfeasible. Thus, testing subsets of configured products is one approach to solving this issue. To reduce the testing effort and obtain better results, alternative methods for testing SPLs are required, such as the combinatorial interaction testing (CIT) technique. Unfortunately, the CIT method produces unscalable solutions for large SPLs with excessive constraints. The CIT method costs more because of feature combinations. The optimization of the various conflicting testing objectives, such as reducing the cost and configuration number, should also be considered. In this article, we proposed a search-based software engineering solution using multi-objective evolutionary algorithms (MOEAs). In particular, the research was applied to different types of MOEA method: the Indicator-Based Evolutionary Algorithm (IBEA), Multi-objective Evolutionary Algorithm based on Decomposition (MOEA/D), Non-dominant Sorting Genetic Algorithm II (NSGAI), NSGAIII, and Strength Pareto Evolutionary Algorithm 2 (SPEA2). The results of the algorithms were examined in the context of distinct objectives and two quality indicators. The results revealed how the feature model attributes, implementation context, and number of objectives affected the performances of the algorithms.

Keywords: search-based software engineering; software product lines; feature models; multi-objective optimization algorithms



Citation: Jamil, M.A.; Nour, M.K.; Alotaibi, S.S.; Hussain, M.J.; Hussaini, S.M.; Naseer, A. Software Product Line Maintenance Using Multi-Objective Optimization Techniques. *Appl. Sci.* **2023**, *13*, 9010. <https://doi.org/10.3390/app13159010>

Academic Editor: Vito Conforti

Received: 11 June 2023

Revised: 30 July 2023

Accepted: 31 July 2023

Published: 6 August 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Software companies have used effective testing techniques such as automated testing to improve the quality of their software products. However, their derived products are selected from a broad product platform, and the testing procedure for a software product line (SPL) can be complicated and expensive. Moreover, the testing process for an SPL is challenging due to the large number of product variants that need to be tested. By producing test cases automatically, this testing procedure can be made more effective and scalable. Software engineering problems can be solved computationally using a meta-heuristic strategy, and this is what search-based software engineering (SBSE) represents [1]. Search-based software testing (SBST) assists in generating optimal test cases in terms of time and cost using metaheuristic techniques [2,3]. Due to the redundancy of test cases, the testing process becomes time-consuming. The various testing optimization methodologies

help to identify issues related to resource utilization and component coordination. Due to the exponentially growing features in the product line, a classification for product sampling approaches that is based on algorithm type and coverage criteria has been proposed [4].

The process of developing a product line is difficult now because there are thousands of variations [5,6]. Using interactive tools and procedures that are appropriate for huge data sets and interrelationships, the intriguing efforts can be resolved [7]. The testing of a software product line is a challenging activity, but it will be optimal if all the products in the product line are configured properly [8]. However, it is complex to generate configurations. SPL testing is a hard process because of the complex composition and design of products [9]. Operating systems are excellent examples of highly configurable systems, such as the Linux kernel. Linux may be set up to run on specific hardware, such as an x86 or x64 processor, but there is also the option of configuring the system's functionality. For instance, it is possible to activate support for a built-in webcam video capture tool that is built-in. The configuration options are limited so that the user cannot select settings that will break the system's consistency. In reality, certain features enable millions of configurations. For instance, a video player product line's feature model (*FM*) with 71 features enables it to configure more than 1.65×10^{13} different products. To reduce a product line's test suites, it will be unable to test every product in an SPL at a reasonable cost. The focus of this article was to optimize a large number of test cases in software product lines. For this, five different multi-objective evolutionary algorithms were selected to optimize the SPL test suite. In this context, we addressed the following research questions in this article:

RQ 1: How will the proposed method, which uses multi-objective optimization (in the case of four objectives optimization), provide the solutions?

RQ 2: Which algorithm, employing the four objectives optimization, is the best fit for producing the optimized results?

The article is significant due to the following achievements (1) a realistic, genuine application of the methodology involves working with both small and large systems, (2) therefore, both types of systems are validated by the proposed approach along the optimization of different objectives, (3) we can say that the proposed technique can be used for different MOEAs with either complicated or simple systems in our experiments, and (4) it is intended to minimize the number of products and the size of test cases.

The rest of the article is organized as follows: Section 2 provides related work on software product line testing and multi-objective algorithms, and Section 3 describes how multi-objective optimization helps to optimize SPL testing. Section 4 presents the method description and testing objectives, and Section 5 describes the experimental setup and results. Lastly, Sections 6 and 7 present the conclusion and future directions of our work.

2. Software Product Line Engineering

Software product line engineering techniques are utilized to make it possible to develop a variety of products quickly, affordably, and with an excellent quality [10]. This methodology uses several features to produce particular systems or products [11]. Throughout the SPL process, the reusable components are combined to generate the desired product.

The product line technique has improved the quality of software [12]. The development life cycle objects, such as requirement specification documents, conceptual diagrams, architectures, reusable components, and testing and maintenance techniques, are merged to meet the process goals [12]. The reusability of components is essential to the product line's methodology. The methodology is divided into two main phases. The first phase is "Domain Engineering", wherein the analysis, design, and implementation of that domain are used to generate core assets. The second phase is "Application Engineering", where customers' unique specifications are used to develop final products [12]. The workflow for software product line engineering is shown in Figure 1. The development of core assets occurs during the domain engineering phase of SPL engineering, whereas their setup takes place during the application engineering phase. Variation points exist that aid in product configuration [5]. When compared to conventional development methods, domain and

application engineering efforts are more successful. To efficiently ensure productivity, the application engineering procedures must operate well [5].

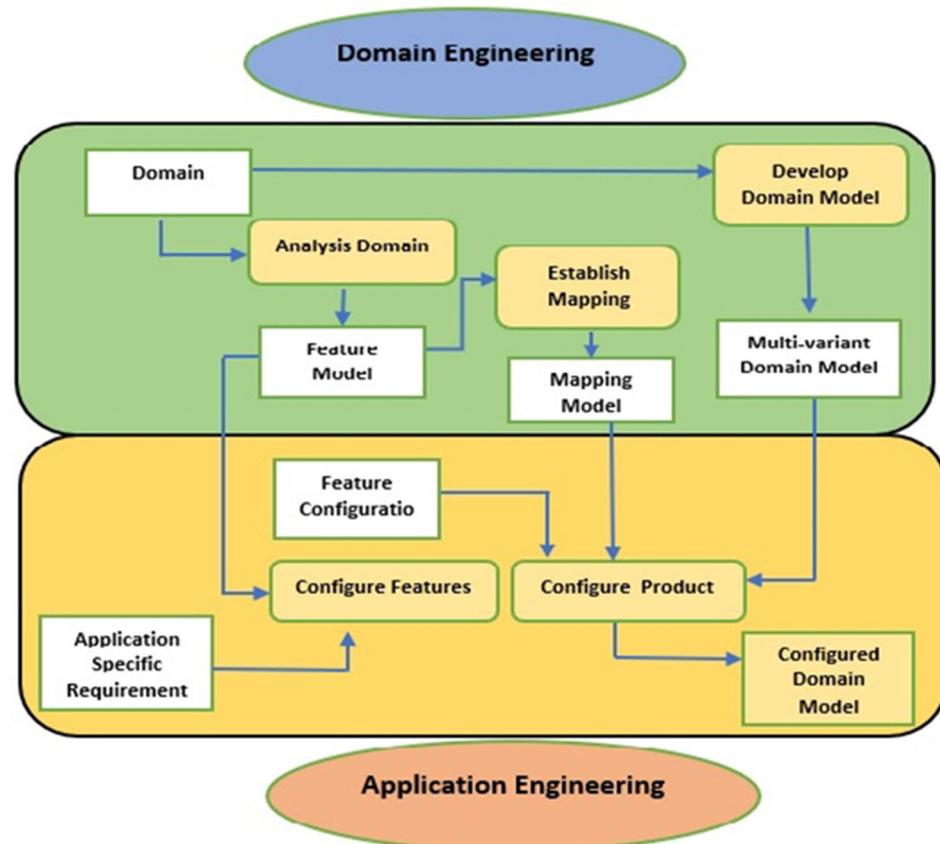


Figure 1. Software product line engineering [13].

3. Related Work

The use of search-based approaches in SPL engineering is not common [1]. SPL testing is improved by using the delta modeling approach, where the deltas calculate the product similarity. Using this technique, the dissimilar product would be tested next. This approach has been evaluated in the automotive domain regarding SPL testing [14]. Similarity-based prioritization has been introduced to choose diverse products concerning features that need to be tested during SPL testing [15]. A genetic-algorithm-based framework using fault integration techniques is employed for test specification reuse. The proposed technique was applied to four different SPL models to evaluate the capability of the framework [16]. Krüger et al. [17] discussed mutation operators, which are appropriate for finding variability faults and enhancing the competence of existing or conventional operators. For the regression testing of an SPL, a retest selection technique has been proposed. This approach relies on delta modeling, which calculates the commonality and variability of the product line variants and different versions of these variants. A solution has been proposed by using a change impact analysis to distinguish between variants and versions of variants where there is a need to retest the changed dependencies in the selection of test cases for re-execution [18].

A hybrid multi-objective optimization algorithm called SMTIBEA was proposed. This technique merges the indicator-based evolutionary algorithm (IBEA) with the satisfiability modulo theories (SMT) solving methodology. The approach was applied to large SPLs to compare the state of the artwork. The proposed approach obtained better results in terms of the expressiveness of constraints [19]. The SPL test list generation scheme, which relies on the Harmony Search (HS) algorithm, called SPL-HS, was recommended. This approach requires the minimum number of test cases for all the features that depend on

the required interaction degree. The obtained results show that the SPL-HS technique can challenge existing SPL testing approaches to generate optimal results [20]. The validation partial configurations approach has been proposed to deliberate different cases such as the selection, attributes, and constraints of features, and this technique was applied with the industry variant tool known as pure::variants. This approach helps to validate the partial configurations for different real-world software product lines [21]. Wang et al. presented their work with a focus on test case minimization [22]. They took advantage of a genetic algorithm with an accumulated function with these determinants, (1) test case numbers, (2) coverage in terms of pairs, and (3) find faults. The test cases prioritization technique was introduced with the addition of a cost function definition. This technique relied on a random search methodology in contrast to a genetic algorithm using a (1 + 1) Evolutionary Algorithm. This methodology evaluation process focused on genetic algorithm configurations assigned different weights. In another research study, they worked on other considerations such as resources and execution costs [23]. Ensan et al. recommended an approach using genetic algorithm inclusion with a composed function to calculate the cost and number of errors [24]. Henard et al. worked on the same approach, using a genetic algorithm with a function to deal with contrary objectives [25]. The configuration of features and selection approaches has also been described [26]. The selection techniques were prescribed by them to work on various objectives. For instance, similar techniques, such as evolution and user preferences, were recommended by [27–29].

The authors mentioned [30,31] their work on the evolution, runtime, and execution of SPLs. The configuration of SPLs has been explicitly discussed by other researchers who have mainly focused on cost, user preferences, and feature model constraint rules [32–36]. Sayyed [37,38] recommended an approach for product line configuration, evaluating distinct multi-objective evolutionary algorithms in terms of different aspects such as product selection, the breached rules in feature models, the utilized features, and the exposed faults during the testing process. The use of multi-objective algorithms generated testing results based on different, defined criteria. For example, Lopez [39] suggested Pareto solutions to describe pairwise coverage. Regardless of this, other researchers have presented genetic algorithms using some aggregated functions. In addition to other factors, the pairwise coverage criteria have basic importance in testing. Henard [40] advocated test data generation for mutation testing. In our work, the same operators were used to originate products in terms of test cases for the testing process.

As a result of the aforementioned assessment of the gaps or weaknesses in the literature, (1) the number of test cases and test suites for the SPL testing procedure has to be decreased, and (2) it can be difficult to optimize multiple objectives simultaneously when dealing with a multi-objective optimization problem. The size of the test cases is one of the key issues in product line testing. Thus, to perform product line testing completely, all the generic components and even all the potential product versions must be tested. In fact, complete testing becomes challenging as the number of product variants increases [3,41]. As a result, the system undergoes more testing as more products are generated. At this stage, the main focus is on minimizing redundant tests in order to reduce the testing effort. The relationship between the produced and derived goods from the same requirements should be taken into account when minimizing test redundancy [3,41].

In an SPL, the generation of configurations is a multi-objective issue. As a result, the activity of SPL testing can be viewed as the development of configurations, which is a multi-objective problem. As a result, it is possible to view the testing of a product line as a multi-objective problem. Although SPL testing using the CIT approach can be a single-objective problem, there is also a need to optimize other objectives, such as minimizing the number of configurations, maximizing the number of features, and minimizing the cost of testing these products [27]. The optimization of many objectives can be difficult though, as achieving one objective may prevent the optimization of other objectives. The results of the implementation of multi-objective evolutionary algorithms (MOEAs) in SPL testing will be used to determine any interventions made by software testers.

This study establishes the rationale for employing MOEA algorithms in SPL testing. The proposed algorithm will help to find more accurate and beneficial results in terms of performance, accuracy, and cost. The suggested work will measure the MOEAs. This research will assist the SPL testing industry in utilizing MOEAs to aid SPL testing processes. In addition, the adopted MOEAs will be competent in setting the precedent for the future generation and selection of test cases to reduce the number of test cases to cope with the time-limited environment. Hence, the ultimate goal of this research is to maximize the coverage of valid features in a product line that needs to be tested. In this work, the SPL testing technique comprises the optimization of multiple objectives; therefore, there may be a need to compromise on some objectives to optimize one objective. Hence, the optimization of conflicting objectives is also considered to be a limitation. The evaluation of configurations can take hours or days; hence, it is computationally expensive. Therefore, discovering other approaches besides the configuration evaluation technique to find faults and robust results rather than the CIT approach is required.

4. Optimization for SPL Testing Using Multi-Objective Evolutionary Algorithms

4.1. Method Description

The different definitions concerning the research problem are outlined as required.

4.1.1. Definition 1 (Feature Model)

A Feature Model is a set of m Boolean features with certain constraints, where a feature model (FM) is expressed as $F = \{f_1, f_2, \dots, f_m\}$, while set $L = \{c_1, \dots, c_n\}$ represents the constraints for the features. A constraint C can be fulfilled by seeing whether the imperatives or constraints are satisfied. Mendonca [42] mentioned that a feature model can be expressed as a Boolean representation, using constraint solvers to analyze a feature model. Such Boolean expressions are defined as the combination of logical clauses. Therefore, a clause is set forth for a feature model, and it literally specifies whether a feature is true (f_j -selected) or false (f'_j -not selected). Hence, to describe an FM , the following Boolean expression can be used.

$$FM = \bigwedge_{i=1}^k \left(\bigvee_{j \in [1, m]} l_j \right), \text{ where } l_j = f_j \text{ or } \bar{f}_j \quad (1)$$

The use of search-based approaches in SPL engineering is not common [1]. SPL testing is improved by using the delta modeling approach, where the deltas calculate the product similarity.

4.1.2. Definition 2 (Configuration)

The features recommended by a product of an SPL are configurations. Therefore, in a configuration, either there will be a selection of particular features or not, while satisfying the rules of the defined constraints. For a product of an SPL, a configuration is considered a set $C = \{\pm f_1, \dots, \pm f_n\}$, where $+f_i$ the feature shows its presence in the FM for the respective product, while $-f_i$ represents the absence of that feature. However, here, it is important that a configuration is valid if the constraints of the FM are satisfied.

4.1.3. Definition 3 (Configuration Suite)

A configuration suite consists of a set of valid configurations and is represented as $CS = \{C_1, \dots, C_m\}$, as each object C_i presents a valid configuration.

4.1.4. Definition 4 (Coverage Criteria)

Coverage Criteria is defined as a set of t-sets covered by the configuration, where the t-set can be defined as a configuration with a certain number of valid and invalid features,

as defined in the last sections. The below ratio is given to describe the pairwise coverage of a configuration suite $CS = \{C_1, \dots, C_m\}$.

$$\frac{\#\bigcup_{i=1}^m T_{t,C_i}}{\#T_{t,FM}} \tag{2}$$

where T_{t,C_i} is the set of t-sets encompassed by the configuration C_i (i.e., the t-sets entailed within the configuration C_i), where $T_{t,FM}$, is indicative of the set of all the possible t-sets of the FM, and where the cardinality of set A is depicted by #A. For t-wise testing, the above coverage ratio is considered to be 1 using classical approaches, because these approaches cover all the t-tests of a feature model [41–44].

For example, the cell phone feature model depicted in Figure 2 can have the following different number of configurations.

$$C_1 = \{+f_1, +f_2, +f_3, +f_4, +f_5, +f_6, -f_7, +f_8, -f_9, -f_{10}, +f_{11}, -f_{12}, -f_{13}\}$$

$$C_2 = \{+f_1, +f_2, +f_3, +f_4, +f_5, -f_6, +f_7, -f_8, +f_9, -f_{10}, +f_{11}, -f_{12}, -f_{13}\}$$

$$C_3 = \{+f_1, -f_2, +f_3, +f_4, +f_5, -f_6, -f_7, -f_8, -f_9, +f_{10}, +f_{11}, -f_{12}, -f_{13}\}$$

$$C_4 = \{+f_1, +f_2, +f_3, +f_4, -f_5, +f_6, -f_7, -f_8, +f_9, -f_{10}, -f_{11}, -f_{12}, -f_{13}\}$$

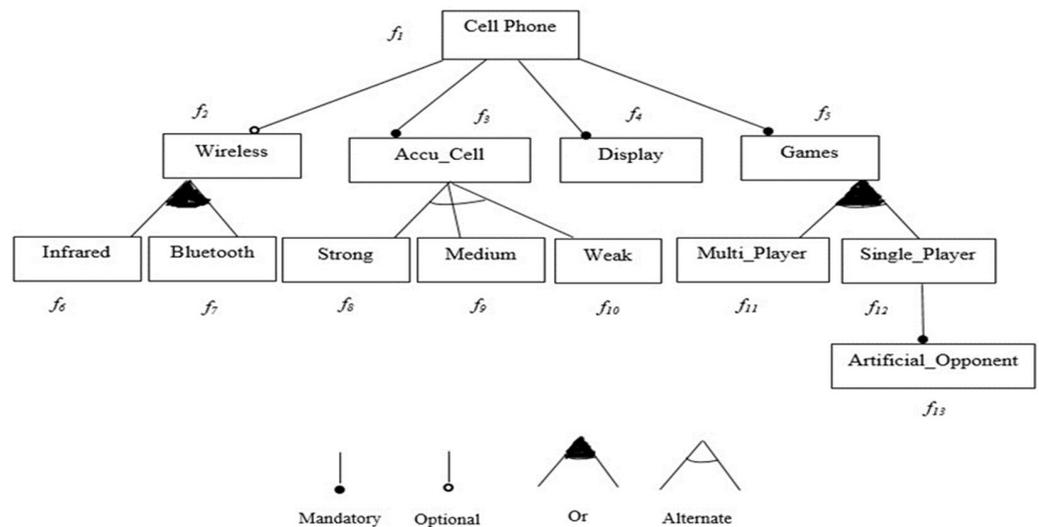


Figure 2. Cell phone feature model (adapted [45]).

4.1.5. Definition 5 (Individual)

An individual is expressed by the set of a configuration suite, i.e., $I = \{C_1, \dots, C_n\}$, as I generates solutions for the problem, while a gene is produced by a valid configuration and the Genetic Algorithm (GA) manages the set of individuals. Hence, the search space of all the possible sets of configurations that are capable of holding configurations in an SPL is acquired.

4.1.6. Definition 5 (Population)

A Population is a set of individuals where a gene is shown by a valid configuration. The genetic algorithm (GA) deals with a set of individuals considered to be a population. Hence, a large search space of sets of valid configurations that exist in an SPL is obtained.

The Materials and Methods should be described with sufficient details to allow others to replicate and build on the published results. An example of the cell phone feature model

is described in Figure 2, which shows a total of 13 features represented by f_1, f_2, \dots, f_{13} . The above Figure 2 shows the cell phone as a root feature of the feature model and this root feature will be included in every product configuration. In this *FM*, the Accu_Cell, Display, and Games features are considered to be mandatory features. These three features would be present in each configuration. There exists an exclusive-or relationship between Accu_Cell and its child features. This means that there would be an explicit inclusion of one out of three features, i.e., Strong, Medium, or Weak.

However, in the case of multi-player and single-player features forming an inclusive-or relationship, which means a valid configuration, there is a selection of at least one feature among two features. An Artificial Opponent is considered to be a mandatory child feature for the Single Player. For the main cell phone parent, the Wireless feature is taken as an optional child feature, therefore, its selection may be optional. The Wireless feature has Infrared and Bluetooth child features and makes an inclusive-or association. This means that, if any configuration contains a Wireless feature, then it must have at least one child feature to be included or selected. Three crosstree constraints persist in the cell phone product line. As long as there is an important selection for a Multi_Player feature that cannot be considered with the Weak feature, hence for this Wireless feature, the selection is mandatory. In the end, for the Bluetooth feature, the Strong feature selection is also compulsory. The set of n products represents the individual population, as exemplified in Figure 3. In that example, every single X contains three products, based on the integer numbers: (1, 4, and 9). A binary vector represents each product, and every element represents a feature of a feature model. The value (0) represents a feature that is not selected, and (1) represents the corresponding feature. In this example, product 1 does not contain the features of Infrared and Artificial Opponent.

Individual A, having 3 products						
1 4 9						
Where each product corresponds to selection (1) or not (0) of each FM feature						
Wireless	Bluetooth	Infrared	Display	Games	Artificial Opponent	
Product 1:						
1	1	0	1	1	0	
Product 4:						
1	0	1	1	1	1	
Product 9:						
1	0	0	1	1	0	

Figure 3. Individual Representation.

To achieve the optimization criteria, it is difficult to identify an SPL configuration that fulfills a number of potentially conflicting objectives. The issue becomes intensified when a configuration suite fulfilling testing goals comprising a set of configurations needs to be generated, which is assumed to be the actual issue that needs to be handled. Constraint-solving techniques and multi-objective evolutionary algorithms are parts of our approach, and to produce configuration suites, these approaches need to be applied in a balanced way so that the fulfillments of four testing objectives can be analyzed at the same time. For SPLs, the configuration generation problem is modeled as a search problem. In the proposed approach, sets of configurations are modeled as individuals and configurations as genes. It is also indicative of the likely operations on the individuals and the objective function. Hence, search-based approaches are enabled to fix the configuration generation problem. To eliminate invalid configurations from the search space, the constraint-solving technique is used.

To provide the maximum size of an individual, there is a need to specify the size of the population. A set of $1, \dots, m$ configurations randomly selected from all the valid configurations towards the feature model (FM) is represented by each individual. The evolution of the population into a new one is considered to be the second step of the approach. The crossover and mutation are the next steps. The individuals are created using mutation and crossover operators to complete the new population until it attains size n . Crossover aims to create two offspring from their selected parents. The user initially specifies a certain probability of mutation taking place in the offspring. Afterward, the fitness of these two offspring is analyzed, and the new population is integrated with these two new individuals. Finally, the new population replaces the existing population and continues into the next generation. Upon the termination of the algorithm, the best fitness value for the individual is obtained. The proposed approach is shown in Figure 4.

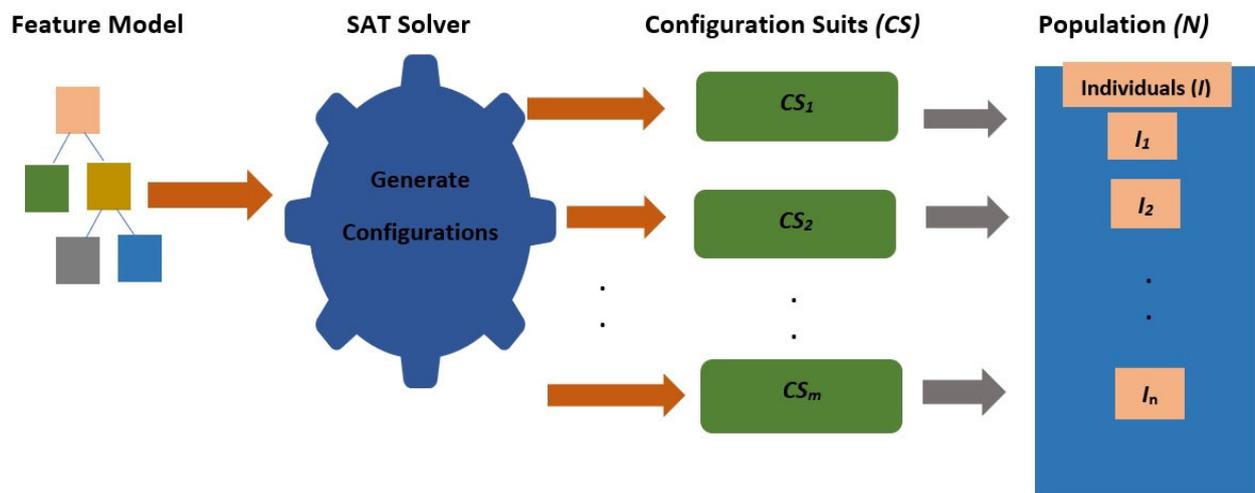


Figure 4. Method Description.

4.2. Objectives Definition

To achieve the optimization criteria, different objectives are defined, which might create conflicts with each other during the optimization process. In the proposed research, the following objectives are considered.

4.2.1. Maximize the Pairwise Coverage

This objective ensures the maximum level of pairwise coverage from the selected products, where the Coverage Criteria is defined as a set of t-sets covered by the configuration, where the t-set can be defined as a configuration with a certain number of valid and invalid features, as defined in the last sections. It is represented as:

$$O1(a) = coverage(a) \quad (3)$$

where the coverage function is used to calculate the number of pairs of features covered by set $a = \{P_1, \dots, P_n\}$.

4.2.2. Minimize the Number of Products

Each product represents a test case. Therefore, the number of products should be minimized. The goal of this objective is to minimize the number of products:

$$O2(a) = cardinality(a) \quad (4)$$

where the cardinality function determines the number of n products $a = \{P_1, \dots, P_n\}$ and it is assumed that a feature exists implicitly at most once in each product. The cardinality of a feature also demonstrates how many times the feature is included in a configuration.

4.2.3. Testing Cost

The testing cost of products is minimized using the following objective function.

$$O3(a) = cost(a) \tag{5}$$

This function makes sure that the testing cost will be minimized with the help of the above-defined objective. The different resources and costs required by each feature need to be tested. In this regard, each feature is assigned a value indicative of an estimate of its testing cost. Therefore, the sum of the cost of its features is presumably equivalent to the cost of testing for one configuration. In other words, if the cost of the feature f_i is symbolized by $cost(f_i)$, then a configuration $C = \{\pm f_1, \dots, \pm f_n\}$ possesses a cost equal to:

$$cost(C) = \sum_{i=1}^n p(i)cost(i), \text{ where } p(i) = \begin{cases} 1 & \text{if } +f_i \\ 0 & \text{if } -f_i \end{cases} \tag{6}$$

The cost of the m configurations is calculated by summing up the cost of a test configuration suite. Hence, the below equation gives the cost of $CS = \{C_1, \dots, C_m\}$:

$$cost(CS) = \sum_{i=1}^m cost(C_i) \tag{7}$$

4.2.4. Maximizing Number of Features

The population is Configuration Suite $\{C_1, C_1 \dots, C_k\}$, which is used to count the number of positive features in a vector of Configuration $C_i = \{+f_1, \dots, f_n, -f_1, \dots, -f_n\}$. The goal of this function is to count features. This objective can be defined as:

$$CS_k = \sum_i^n \sum_j^m \lambda(PF(\sqsupset, j)) \tag{8}$$

where $\lambda(PF(i, j)) = 1$, if $\sum_i^n (PF(i, j)) > 1$, else = 0.

The CS represents the configuration suite, the PF shows the product features, and 1 represents one whole product with binding features.

5. Experiments

In this section, we discuss the framework adopted, the data collection procedure, and the parameter settings.

5.1. Framework Adopted

This section describes how the experiments were designed and conducted: SPL feature models, algorithm implementation, and quality indicators for the comparison of the algorithms. The approach employed was similar to that adopted in our recent research work [46]. For evaluation and implementation, five multi-objective algorithms were selected: IBEA [47], MOEA/D [48], NSGAII [49], NSGAIII [50], and SPEA2 [51]. There exists a free and open-source Java library framework called “The MOEA Framework”. It is used for both single- and multi-objective optimization algorithms. The MOEA Framework aids genetic algorithms, grammatical evolution, particle swarm optimization, genetic programming, and differential evolution [52]. This framework consists of operators and solutions for optimization problems, and it also implements quality indicators such as hypervolume.

The selected algorithms were configured and re-executed to generate and report the results. As mentioned above in Equation (1), the Boolean formula of a feature model was encoded into a SAT solver to resolve whether a configuration was valid or invalid for a feature model. The formula was applied as a conjunction form for all the constraints and evaluated to true (valid) values for the constraints satisfied by the solver.

5.2. Data Collection

The six feature models were selected with dissimilar sizes and obtained from Software Product Line Tools (SPLIT) [9]. Table 1 displays the various attributes of these selected feature models. The six feature models were considered for four objectives of optimization, with smaller-, medium-, and larger-size features to analyze the results. The feature model characteristics are shown in Table 1. The first column shows the different feature models, the second represents the number of features, and the third and fourth columns show the number of configurations and the number of valid pairs, respectively.

Table 1. Feature model attributes.

Feature Model	Features	Configurations	Number of Pairs
Counter Strike	24	18,176	833
SPL SimulES, PnP	41	6912	2592
Smart Homev2.2	60	3.87×10^9	6189
Video Player	71	4.5×10^{13}	7528
Model Transformation	88	1.65×10^{13}	13,139
Coche Ecologico	94	2.32×10^7	11,075

5.3. Parameters Selection

The parameter settings were adopted before initiating the experiments. The fixed parameters were finalized at the beginning to obtain a constant result and provide the possibility of comparing the efficiency of the algorithms involved in the experiments. The selected parameters were the size of the population, the maximum number of evaluations, and the sizes of the *FM*, crossover, and mutation rates, as shown in Table 2.

Table 2. Parameter Settings.

Parameter	Values
Population Size	200
Number of Generations	500
Crossover Rate	60%
Mutation Rate	30%

6. Results and Discussion

We performed experiments that relied on the experimental methodology discussed in the previous sections. First, the results of the Pareto fronts are presented with tuned parameters for the selected SPLIT feature model configurations.

6.1. Pareto Front Solutions

The results of the proposed research are shown and analyzed in this section. The non-dominated solutions are presented in Table 3, based on the Pareto dominance concepts [53]. Table 3 presents the main results generated by each algorithm and feature model for the multi-objective optimization techniques. Column 2 shows the total number of solutions in the global pareto front PF_{true} , formed by the non-dominated solutions generated by the executions of all five algorithms. However, the other columns present the number of non-dominated solutions generated by an individual algorithm known as $PF_{contribution}$, which are present in PF_{true} , and the values in parentheses show the percentage of the non-dominated solutions of each algorithm as compared to the global solutions PF_{true} .

Table 3. Pareto fronts with four objectives solutions.

Feature Models	Algorithms					
	PF_{true}	IBEA	MOEAD	NSGAII	NSGAIII	SPEA2
		$PF_{contribution}$				
Counter-Strike	465	13 (2.81%)	122 (26.23%)	24 (5.16%)	109 (23.44%)	197 (42.36%)
SPL SimulES, PnP	456	10 (2.19%)	138 (30.26%)	13 (2.85%)	97 (21.27%)	198 (43.42%)
Smart Home v2.2	499	8 (1.60%)	144 (28.85%)	23 (4.60%)	124 (24.84%)	200 (40.08%)
Video Player	456	8(1.75%)	124 (27.19%)	16 (3.5%)	108 (23.68%)	200 (43.85%)
Model Transformation	510	17 (3.4%)	141 (27.64%)	19 (3.72%)	133 (26.07%)	200 (39.21%)
Coche Ecologico	490	27 (5.51%)	123 (25.10%)	11 (2.24%)	129 (26.32%)	200 (40.81%)

The preliminary analysis in Table 3 shows that the IBEA and NSGAII algorithms generated a certain number of solutions in PF_{true} in consideration of the used objectives. All the algorithms used in this research found solutions for all the feature models in all runs. The IBEA algorithm generated the minimum number of solutions for the SmartHome v2.2 and Video Player feature models. However, in the case of the other feature models, where more solutions originated from IBEA, the larger feature model Coche Ecologico generated a greater number of non-dominated solutions, resulting in 27 solutions as compared to the other feature models. In the case of NSGAII, a diversity of solutions was observed for the small-size feature model Counter-Strike; it originated from 24 solutions, while in the case of the larger-size feature model Coche Ecologico, it generated the minimum number of solutions, i.e., 11. Similarly, to observe the results of NSGAIII, MOEA/D, and SPEA2, these three algorithms generated a good number of non-dominated solutions for all the feature models. Table 3 simply analyzes that all the algorithms worked effectively to optimize the four objectives problem.

Figure 5 shows the distinct percentage values for the non-dominated solutions generated by the five different MOEAs. It is obvious from Figure 4 that the SPEA2 algorithm generated the maximum percentages of solutions in all six selected feature models. However, MOEA/D was also able to generate a good amount of percentage solutions for the FMs mentioned in Table 3. For MOEA/D, the solution percentages originated for Counter-Strike, SPL SimulES PnP, Smart Homev2.2, Video Player, Model Transformation, and Coche Ecologico were 26.23, 30.26, 28.85, 27.19, 27.64, and 25.10, respectively. Figure 5 also describes that NSGAIII contributed well to producing good percentage values of non-dominated solutions, while the IBEA and NSGAII percentage solutions were not satisfactory as compared to the results of the other MOEAs in terms of percentage.

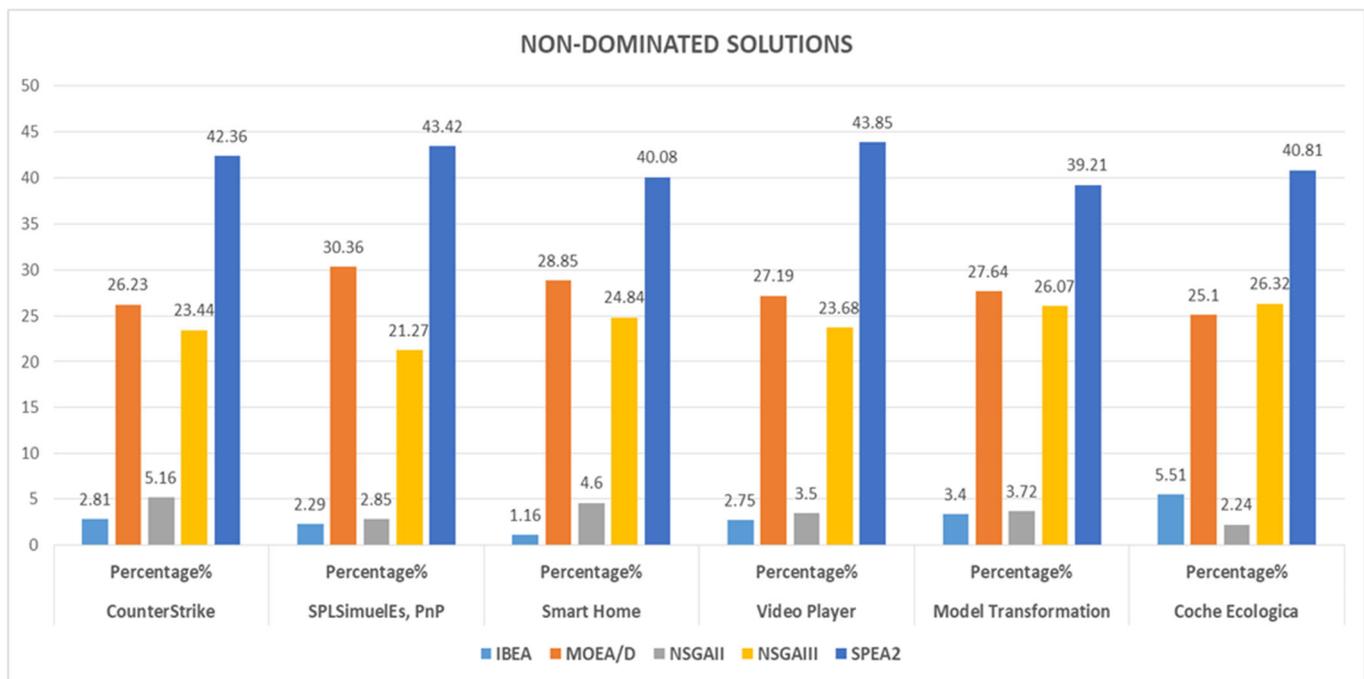


Figure 5. Non-dominated solution values for different feature models.

6.2. Results Generated by Quality Indicators

The evaluation process for the results of the MOEAs was carried out by using quality multi-objective indicators and a statistical analysis, as discussed in [54,55]. In this research work, two quality indicators were selected to evaluate the results: Hypervolume [56] and Spacing [57]. There is a possibility that the efficiency of the MOEAs may have had an effect on the four objectives, therefore, to measure the better performance of the MOEAs, these two quality indicators were utilized to determine the spread, convergence, and smaller space for the non-dominated solutions generated by the algorithms regarding the Pareto front [55,58].

Table 4 represents the hypervolume and spacing indicators with their average and standard deviation values, respectively, for the 30 $PF_{contribution}$ sets generated by each algorithm. To establish a quantitative analysis of the results, the above-mentioned quality indicators were utilized. These quality metrics helped to generate good-quality values [56]. The first column of Table 4 represents the hypervolume and spacing indicators, while the other columns represent the feature models and different algorithms to calculate the average and standard deviation values using the mentioned indicators. The best values are represented in bold in Table 4.

For hypervolume, higher values are considered the best, while minimum values are taken into account as the best for spacing. To analyze the hypervolume values for CounterStrike *FM*, it is discovered from Table 4 that the MOEA/D and SPEA2 algorithms originated the best average and standard deviation values (0.7094 and 0.0129) and (0.7022 and 0.0165), respectively. Comparatively, the NSGAIII results are also considered as better (0.6498 and 0.0113). However, in the cases of IBEA (0.2826 and 0.1162) and NSGAI (0.6029 and 0.0234), the generated hypervolume and spacing values are considered to be less than the number of solutions.

For the SPL SimuelEs, PnP *FM*, it can be deduced from Table 4 that the NSGAIII values are not desirable in terms of average and standard deviation (0.5413 and 0.0176), and the same behavior is displayed by IBEA and NSGAI, showing values of (0.2698 and 0.0954), and (0.4659 and 0.0314), respectively. MOEA/D generated the best values (0.6524 and 0.0150) in contrast to the other MOEAs. The values of SPEA2 are sufficient for creating the optimal values of the average and standard deviation (0.6390 and 0.0155).

Table 4. Indicators of hypervolume (HV) and spacing values for four objectives.

Indicator	Feature Models	IBEA		MOEAD		NSGAI		NSGAIII		SPEA2	
		Average	St. D	Average	St. D	Average	St. D	Average	St. D	Average	St. D
HV	Counter Strike	0.2826	0.1162	0.7094	0.0129	0.6029	0.0234	0.6498	0.0113	0.7022	0.0165
	SPL SimulES, PnP	0.2698	0.0954	0.6524	0.0150	0.4659	0.0314	0.5413	0.0176	0.6390	0.0155
	Smart Home v2.2	0.3460	0.1105	0.6885	0.0160	0.5742	0.0353	0.6052	0.0300	0.6843	0.0198
	Video Player	0.3659	0.0969	0.6961	0.0155	0.5795	0.0108	0.6085	0.0262	0.6864	0.0193
	Model Transformation	0.3600	0.0771	0.6009	0.0146	0.4677	0.0256	0.5276	0.0210	0.6136	0.0210
	Coche Ecologico	0.3257	0.0802	0.4868	0.0048	0.3227	0.0107	0.4541	0.0329	0.5025	0.0013
Spacing	Counter Strike	0.1138	0.0543	0.0329	0.0033	0.0496	0.0115	0.0316	0.0026	0.0157	0.0017
	SPL SimulES, PnP	0.0713	0.0364	0.0208	0.0019	0.0521	0.0121	0.0295	0.0026	0.0113	0.0011
	Smart Home v2.2	0.0753	0.0330	0.0302	0.0024	0.0474	0.0102	0.0342	0.0037	0.0126	0.0086
	Video Player	0.0855	0.0430	0.0296	0.0028	0.0502	0.0108	0.0328	0.0046	0.0127	0.0015
	Model Transformation	0.0527	0.0112	0.0262	0.0021	0.0412	0.0081	0.0292	0.0024	0.0102	0.0008
	Coche Ecologico	0.0417	0.0173	0.0217	0.0016	0.0720	0.0218	0.0179	0.0030	0.0095	0.0013

In the case of the Smart Homev2.2 *FM*, nearly the same average values with a minor difference in standard deviation (0.6885 and 0.0160) and (0.6843 and 0.0198) can be generated by both the MOEA/D and SPEA2. NSGAIII remained deliberate in terms of producing good result values (0.6052 and 0.0300) for the Smart Homev2.2 *FM*. Nonetheless, the performances of IBEA and NSGAI are observed to be unsatisfactory, as they generated (0.3460 and 0.1105) and (0.5742 and 0.0353) values, respectively.

While considering the results of the Video Player *FM*, the SPEA2 and MOEA/D both produced almost the same average values with an insignificant difference in standard deviation, (0.6961 and 0.0155) and (0.6864 and 0.0193). NSGAIII stayed deliberate concerning producing desirable result values (0.6085 and 0.0262) for the Smart Homev2.2 *FM*. On the other hand, IBEA and NSGAI delivered unsatisfactory performances and generated (0.3659 and 0.0969) and (0.5795 and 0.0108) values for the average and standard deviation, respectively.

After analyzing the results of the Model Transformation *FM*, the MOEA/D and SPEA2 both generated approximately the same average values but with a minor difference in standard deviation values, (0.6009 and 0.0146) and (0.6136 and 0.021). NSGAIII was found to originate better result values (0.5276 and 0.021) for the Smart Homev2.2 *FM*. On the contrary, an unacceptable performance was demonstrated by IBEA and NSGAI, and the generated values were (0.36 and 0.0771) and (0.4677 and 0.0256), respectively.

When the results of Coche Ecologico are investigated, it is found that approximately the same average values were generated by both the MOEA/D and SPEA2, with a slight change in standard deviation, (0.4868 and 0.0048) and (0.5025 and 0.0013). Moreover, NSGAIII yielded good result values (0.4541 and 0.0329) for the Smart Homev2.2 *FM*. Though IBEA and NSGAI illustrated poor performances, they generated (0.3257 and 0.0802) and (0.3227 and 0.0107) values, respectively.

In Table 4, it is examined that, for all the feature models, the SPEA2 algorithm generated the best average and standard deviation values using the spacing metric. However, the MOEA/D and NSGAIII algorithms also generated good values for the average and standard deviation.

To conclude, in Table 4, it is determined that SPEA2 and MOEA/D can produce better values. It is worth mentioning that NSGAIII is also good at producing a desirable result for most of the feature models. Nevertheless, unsatisfactory performances are revealed by IBEA and NSGAI as compared to the other three algorithms.

6.3. Fitness Values for Four Objectives Optimization

Table 5 presents the fitness values of the solutions for the four objectives mentioned in the above section. The fitness values of the four objectives are selected from the results of the experiments. The first column of Table 5 shows the feature models, and the other columns represent the MOEA algorithms with their four objective values.

Table 5. Fitness values of solutions with objectives (*O1*, *O2*, *O3*, and *O4*).

Feature Models	Algorithms				
	IBEA	MOEAD	NSGAI	NSGAIII	SPEA2
Counter Strike	<i>O1</i> (1, 0.010, 0.086, 0.291)	<i>O1</i> (1, 0.010, 0.623, 0.666)	<i>O1</i> (1, 0.010, 0.666, 0.75)	<i>O1</i> (1, 0.010, 0.537, 0.666)	<i>O1</i> (1, 0.010, 0.709, 0.791)
	<i>O2</i> (1, 0.010 , 0.086, 0.291)	<i>O2</i> (1, 0.010 , 0.623, 0.666)	<i>O2</i> (1, 0.010 , 0.666, 0.75)	<i>O2</i> (1, 0.010 , 0.537, 0.666)	<i>O2</i> (1, 0.010 , 0.709, 0.791)
	<i>O3</i> (1, 0.010, 0.086 , 0.291)	<i>O3</i> (1, 0.010, 0.107 , 0.291)	<i>O3</i> (1, 0.010, 0.107 , 0.291)	<i>O3</i> (1, 0.010, 0.107 , 0.291)	<i>O3</i> (1, 0.010, 0.107 , 0.291)
	<i>O4</i> (0.274, 0.030, 0.365, 1)	<i>O4</i> (0.271, 0.071, 0.136, 1)	<i>O4</i> (0.228, 0.051, 0.276, 1)	<i>O4</i> (0.238, 0.051, 0.219, 1)	<i>O4</i> (0.260, 0.051, 0.198, 1)
SPL SimulES, PnP	<i>O1</i> (1, 0.010, 0.358, 0.593)	<i>O1</i> (1, 0.010, 0.312, 0.468)	<i>O1</i> (1, 0.010, 0.343, 0.562)	<i>O1</i> (1, 0.010, 0.458, 0.625)	<i>O1</i> (1, 0.010, 0.664, 0.656)
	<i>O2</i> (1, 0.010 , 0.358, 0.593)	<i>O2</i> (1, 0.010 , 0.312, 0.468)	<i>O2</i> (1, 0.010 , 0.343, 0.562)	<i>O2</i> (1, 0.010 , 0.458, 0.625)	<i>O2</i> (1, 0.010 , 0.664, 0.656)
	<i>O3</i> (1, 0.010, 0.305 , 0.05)	<i>O3</i> (1, 0.010, 0.229 , 0.50)	<i>O3</i> (1, 0.010, 0.236 , 0.50)	<i>O3</i> (1, 0.010, 0.244 , 0.468)	<i>O3</i> (1, 0.010, 0.236 , 0.50)
	<i>O4</i> (0.625, 0.030, 0.419, 1)	<i>O4</i> (0.638, 0.122, 0.279, 1)	<i>O4</i> (0.654, 0.030, 0.358, 1)	<i>O4</i> (0.669, 0.408, 0.328, 1)	<i>O4</i> (0.647, 0.071, 0.312, 1)
Smart Home v2.2	<i>O1</i> (1, 0.010, 0.419, 0.045)	<i>O1</i> (1, 0.010, 0.389, 0.483)	<i>O1</i> (1, 0.010, 0.551, 0.583)	<i>O1</i> (1, 0.010, 0.507, 0.583)	<i>O1</i> (1, 0.010, 0.441, 0.466)
	<i>O2</i> (1, 0.010 , 0.419, 0.045)	<i>O2</i> (1, 0.010 , 0.389, 0.483)	<i>O2</i> (1, 0.010 , 0.551, 0.583)	<i>O2</i> (1, 0.010 , 0.507, 0.583)	<i>O2</i> (1, 0.010 , 0.441, 0.466)
	<i>O3</i> (1, 0.010, 0.257 , 0.366)	<i>O3</i> (1, 0.010, 0.069 , 0.20)	<i>O3</i> (1, 0.010, 0.077 , 0.216)	<i>O3</i> (1, 0.010, 0.036 , 0.133)	<i>O3</i> (1, 0.010, 0.106 , 0.216)
	<i>O4</i> (0.264, 0.306, 0.395, 1)	<i>O4</i> (0.161, 0.061, 0.354, 1)	<i>O4</i> (0.178, 0.071, 0.302, 1)	<i>O4</i> (0.190, 0.071, 0.334, 1)	<i>O4</i> (0.239, 0.071, 0.248, 1)
Video Player	<i>O1</i> (1, 0.010, 0.437, 0.605)	<i>O1</i> (1, 0.010, 0.427, 0.619)	<i>O1</i> (1, 0.010, 0.558, 0.690)	<i>O1</i> (1, 0.010, 0.540, 0.633)	<i>O1</i> (1, 0.010, 0.601, 0.676)
	<i>O2</i> (1, 0.010 , 0.437, 0.605)	<i>O2</i> (1, 0.010 , 0.427, 0.619)	<i>O2</i> (1, 0.010 , 0.558, 0.690)	<i>O2</i> (1, 0.010 , 0.540, 0.633)	<i>O2</i> (1, 0.010 , 0.601, 0.676)
	<i>O3</i> (1, 0.010, 0.234 , 0.450)	<i>O3</i> (1, 0.010, 0.128 , 0.352)	<i>O3</i> (0.801, 0, 0.156 , 0.521)	<i>O3</i> (0.772, 0.010, 0.161 , 0.535)	<i>O3</i> (1, 0.010, 0.163 , 0.380)
	<i>O4</i> (0.128, 0.061, 0.412, 1)	<i>O4</i> (0.174, 0.051, 0.323, 1)	<i>O4</i> (0.206, 0.040, 0.330, 1)	<i>O4</i> (0.279, 0.036, 0.335, 1)	<i>O4</i> (0.214, 0.040, 0.301, 1)
Model Transformation	<i>O1</i> (1, 0.010, 0.534, 0.579)	<i>O1</i> (1, 0.010, 0.327, 0.454)	<i>O1</i> (1, 0.010, 0.415, 0.50)	<i>O1</i> (1, 0.010, 0.372, 0.477)	<i>O1</i> (1, 0.010, 0.461, 0.534)
	<i>O2</i> (1, 0.010 , 0.534, 0.579)	<i>O2</i> (1, 0.010 , 0.327, 0.454)	<i>O2</i> (1, 0.010 , 0.415, 0.50)	<i>O2</i> (1, 0.010 , 0.372, 0.477)	<i>O2</i> (1, 0.010 , 0.461, 0.534)
	<i>O3</i> (1, 0.010, 0.193 , 0.363)	<i>O3</i> (1, 0.010, 0.161 , 0.318)	<i>O3</i> (1, 0.010, 0.170 , 0.340)	<i>O3</i> (1, 0.010, 0.170 , 0.284)	<i>O3</i> (1, 0.010, 0.179 , 0.284)
	<i>O4</i> (0.211, 0.051, 0.393, 1)	<i>O4</i> (0.021, 0.256, 0.404, 1)	<i>O4</i> (0.165, 0.071, 0.371, 1)	<i>O4</i> (0.224, 0.051, 0.342, 1)	<i>O4</i> (0.158, 0.081, 0.319, 1)
Coche Ecologico	<i>O1</i> (1, 0.010, 0.251, 0.563)	<i>O1</i> (1, 0.010, 0.398, 0.659)	<i>O1</i> (1, 0.010, 0.310, 0.574)	<i>O1</i> (1, 0.010, 0.310, 0.574)	<i>O1</i> (1, 0.010, 0.316, 0.617)
	<i>O2</i> (1, 0.010 , 0.251, 0.563)	<i>O2</i> (1, 0.010 , 0.398, 0.659)	<i>O2</i> (1, 0.010 , 0.310, 0.574)	<i>O2</i> (1, 0.010 , 0.310, 0.574)	<i>O2</i> (1, 0.010 , 0.316, 0.617)
	<i>O3</i> (1, 0.010, 0.202 , 0.563)	<i>O3</i> (1, 0.010, 0.156 , 0.510)	<i>O3</i> (1, 0.010, 0.156 , 0.521)	<i>O3</i> (1, 0.010, 0.156 , 0.521)	<i>O3</i> (1, 0.010, 0.156 , 0.50)
	<i>O4</i> (0.143, 0.112, 0.268, 1)	<i>O4</i> (0.180, 0.816, 0.251, 1)	<i>O4</i> (0.094, 0.173, 0.275, 1)	<i>O4</i> (0.094, 0.173, 0.275, 1)	<i>O4</i> (0.149, 0.102, 0.255, 1)

The five algorithms, IBEA, MOEA/D, NSGAI, NSGAIII, and SPEA2, generated fitness values for each objective. Objective one is represented by *O1*, which aimed to maximize the pairwise coverage. The second objective, i.e., the minimization of the product number, is shown by *O2*. *O3* describes objective three, which was utilized to reduce the testing cost. The fourth objective was to maximize the richness of the features and is represented by *O4*. The bold values represent the best-optimized values in comparison to the other objective values. To accomplish the maximum fitness values of the *O1* objective (pairwise coverage), the other three objectives, *O2* (minimize the number of products), *O3* (testing cost), and *O4* (richness of features), have to be compromised, as shown in Table 5. To follow the same approach to obtain the best minimum fitness values for objective *O2*, the other three objectives, *O1*, *O3*, and *O4*, need to be settled. The relevant method would be adopted to obtain the best fitness values for objectives *O3* and *O4*. The best fitness values for the three objectives are shown in bold in Table 5. Besides the evolution of the objectives, the normalized objective values over the generations of the algorithms are presented. Since all four objectives are transformed into the optimization of the problems. It is noted that our approach is not limited to this balance. Thus, the tester or user may set a different balance according to their needs to optimize the testing process of the software product line.

Table 6 provides the analysis through the MOEAs, with the optimal results using both indicators against each feature model. Table 6’s inferences are determined from the results of Table 4.

Table 6. Better MOEA algorithm concerning indicators (HV and spacing).

Feature Models	Quality Indicators	
	Hyper Volume	Spacing
Counter Strike	MOEA/D, SPEA2, NSGAIII	MOEA/D, SPEA2, NSGAIII
SPL SimulES, PnP	MOEA/D, SPEA2	MOEA/D, SPEA2, NSGAIII
Smart Home v2.2	MOEA/D, SPEA2, NSGAIII	MOEA/D, SPEA2, NSGAIII
Video Player	MOEA/D, SPEA2, NSGAIII	MOEA/D, SPEA2
Model Transformation	MOEA/D, SPEA2, NSGAIII	MOEA/D, SPEA2, NSGAIII
Coche Ecologico	MOEA/D, SPEA2, NSGAIII	SPEA2, NSGAIII,

For the Counter-Strike feature model, it is explored that the MOEA/D, SPEA2, and NSGAIII algorithms originated the best values using the hypervolume and spacing indicators. However, in the case of SPL SimulES, PnP, it is observed that MOEA/D and SPEA2 were the best algorithms for generating the optimal results using hypervolume.

However, using the spacing indicator for the SPL SimulES, PnP *FM*, the MOEA/D, SPEA2, and NSGAIII algorithms were capable of producing better values. Similarly, in the case of Smart Homev2.2 *FM*, MOEA/D, SPEA2, and NSGAIII were the best algorithms for generating the optimized results employing hypervolume and spacing. For the Video Player *FM*, the better values using hypervolume originated from MOEA/D, SPEA2, and NSGAIII. However, for the Video Player feature model, better results were obtained only by MOEA/D and SPEA2 using spacing, while in the case of the Model Transformation *FM*, with the same results as those of Smart Homev2.2, the MOEA/D, SPEA2, and NSGAIII algorithms were the best value generators using hypervolume and spacing.

In Table 6, it is examined that, for the Coche Ecologico *FM*, MOEA/D, SPEA2, and NSGAIII had the best values using hypervolume, but after utilizing the spacing, only the MOEA/D and SPEA2 algorithms produced good results. To conclude, in Table 6, it is shown that the best optimal results were generated by MOEA/D, SPEA2, and NSGAIII in most of the feature models.

6.4. Discussion of Results and Answers to Research Questions

It is considered that the solutions generated by every algorithm depended on the related fitness values. The solutions associated with the best results are analyzed to complete the SPL testing assessment, which is also the overall objective of the research. The ability to generate optimal values for four objectives is depicted by the findings revealed in the previous tables. The outcomes of the experiments have the potential to identify better solutions for all the objectives under consideration. Although some disparities may not be statistically meaningful, it can be seen that the approach is a settlement among the conflicting objectives. Hence, due to certain parameters, this is likely to compromise the four objectives. It is worth indicating that the aforementioned results are accomplished by the proposed approach over a reasonable number of generations (500 generations).

Different parameters can have an effect on minimizing the number of products, which could be the answer to the first question. To find optimal solutions, the quality testers can gain insight through this direction as to why the testing optimization techniques were adopted. However, there are some considerable cases where the system reliability is higher than the justification for the test cases. This is the benefit of our proposed methodology, which can produce various solutions in terms of coverage as well. The choice would be in the hands of the tester in terms of adopting a better MOEA for a good performance. Single-objective algorithms produce one solution, which may not be an ideal selection in our case, where multiple objectives need to be optimized. The testers can define the priorities for selecting the values of different operators, such as the crossover rate and mutation rate, to affect the experiment results of different product lines.

The best solutions are identified to assess each feature model. The quality indicators from the optimization field, such as hypervolume and spacing, are also utilized.

In consideration of Research Question 2, the results have a clear impact on the performance of the algorithms with the four objectives optimization, as discussed in detail in the different tables. In this section of four objectives, the versatility of the generated results is demonstrated. However, it is determined that, for all the SPL feature models, IBEA generated the worst results in terms of optimization. MOEAD, SPEA, and NSGAIII performed well in all cases of feature models with small, medium, and larger sizes.

Table 4 presents an analysis of the percentage of non-dominated solutions generated by the different MOEAs. Again, SPEA2 surpassed and originated the maximum percentage of non-dominated solutions for all the *FMs*. Similarly, MOEA/D and NSGAIII also produced a better percentage of solutions. However, the numbers of solution percentages in the cases of IBEA and NSGAII were not comparatively found to be good and generated a lower percentage of non-dominated solutions.

According to Table 5, reasoning with the average and standard deviation results of the hypervolume and spacing quality indicators obliged in the proposed approach, SPEA2 also achieved the best results using the hypervolume and spacing quality indicators. This means the SPEA2 algorithm provided more optimal solutions to the tester, with various and ultimate values of fitness using statistical methods such as average and standard deviation. This indicates that the MOEA/D and SPEA2 values were more optimal.

Table 6 represents the best fitness values regarding the different MOEAs. It can be observed that, to obtain the best value for a particular objective, one may need to compromise on the other objectives' values. Therefore, the proposed approach can generate the optimal fitness values to optimize the four objectives of maximizing the pairwise coverage, reducing the number of products, minimizing the testing cost, and maximizing the feature richness.

It is apparent from the results mentioned in the different tables that three algorithms (SPEA2, MOEA/D, and NSGAIII) accomplished better results. Nevertheless, IBEA's and NSGAII's performances in terms of originating optimal values were not satisfactory. All the algorithms, except IBEA and NSGAII, tried to achieve maximum coverage while minimizing the number of test cases. In future work, IBEA and NSGAII will be investigated more, as tweaking the parameter values may help to generate good results.

To sum up, three MOEA algorithms (MOEA/D, SPEA2, and NSGAIII) can generate optimal solutions for the four-objective optimization approach when compared with single-objective approaches.

Figures 6 and 7 represent the global and local Pareto fronts of the Counter Strike feature model only using five multi-objective evolutionary algorithms (IBEA, MOEA/D, NSGAII, NSGAIII, and SPEA2) in the case of the four-objective optimization approach. Here, only three objective points are represented due to the unavailability of a four-axis representation. The figures show the non-dominated Pareto fronts solutions (with red circled blue points) obtained after the experiments. The Pareto front (*PF*) represents the subset of potential solutions to the MOO issue that are not dominated by the other solutions. These solutions represent the optimal test suites in terms of the SPL testing optimization process. The testers would be able to analyze which test suites should be selected to improve the testing process in their software organizations. This will result in a variety of options, from which the tester may select the best one depending on the performance and requirements.

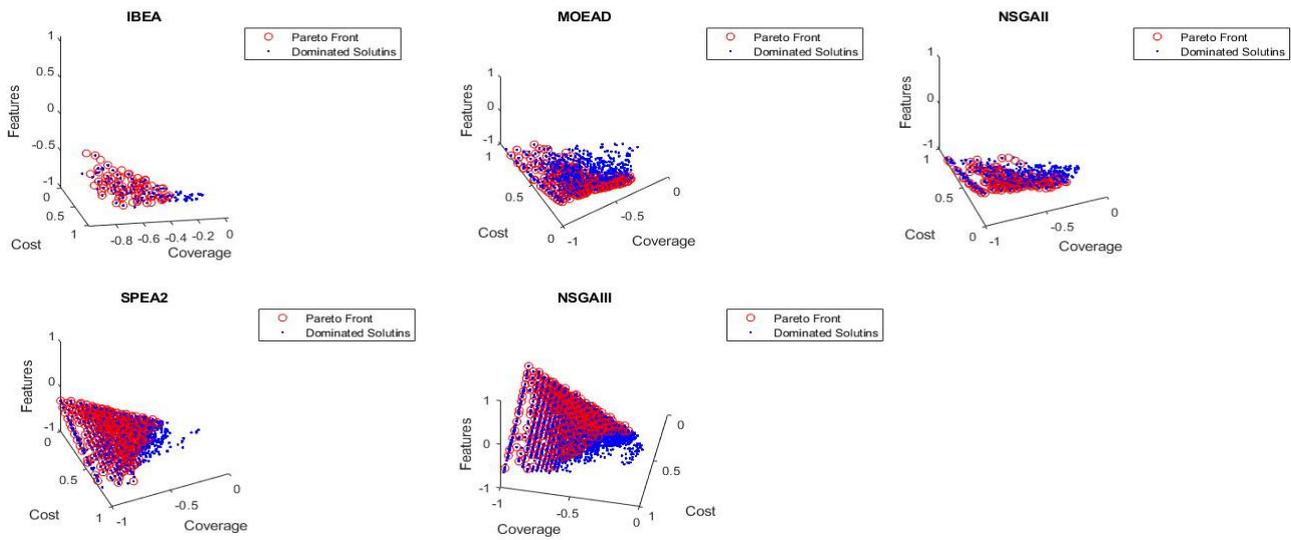


Figure 6. Counter-Strike FM Pareto Fronts (Four Objectives) using Five MOEAs.

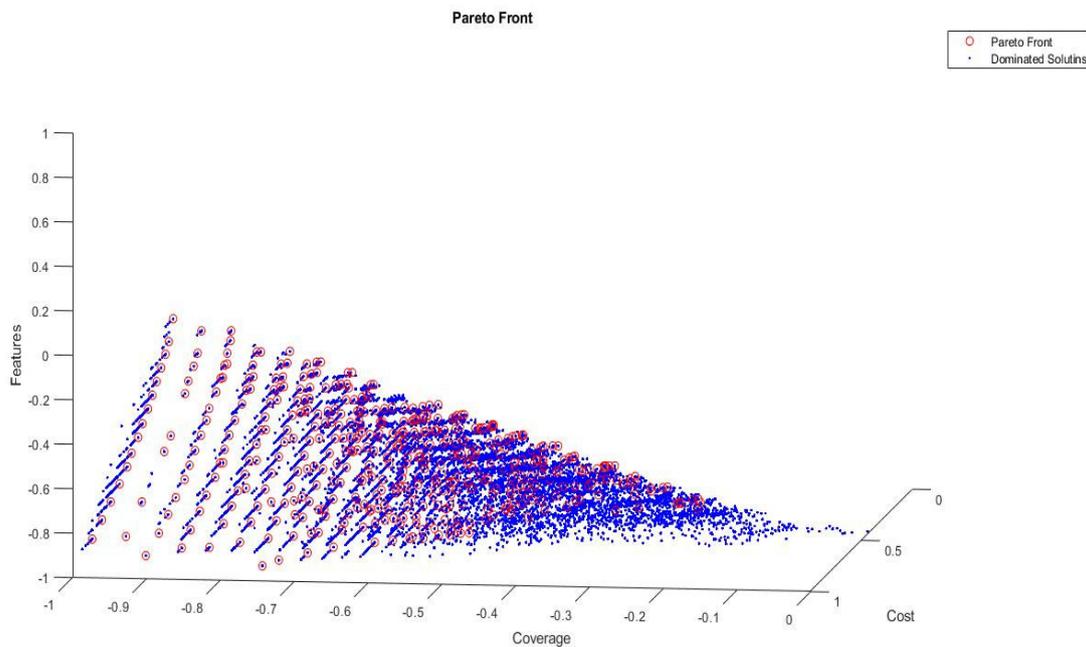


Figure 7. Counter-Strike FM Pareto Fronts Global (Four Objectives).

7. Conclusions

This section will discuss the findings of the experiments and evaluations that were performed. Moreover, the results for the four objectives of the experiments led to the answering of the questions of the proposed research. The advantage over the single-objective approach was one achievement observed in the researcher’s work during the evaluation [24]. The results generated were varied, containing good solutions using multi-objective algorithms. As per the testing goals, there is a possibility that the tester may prioritize a single objective at the end because they need to choose the extreme points or solutions.

In the case of the four objectives, the comparison of the experiments shows that it did not change the cost of the required number of products. Although there were constraints on the testing activities, for example problems related to contracts and development, MOEA/D and SPEA2 can perform better in contrast to the other MOEAs. In this manner, a diversity of solutions will be obtained, and the tester will be able to choose the best one based on the

performance and requirements. In a situation where fast execution is required, SPEA2 is the relatively right choice, as *FMs* have a maximum number of products. SPEA2 should be used in the interest of the tester, as the tester is more inclined towards the best tradeoffs of the solution, as it has a higher number of non-dominated solutions.

7.1. Practical Implications

Different multi-objective algorithms were utilized for the implementation of this approach. The algorithms used within the framework were IBEA, MOEA/D, NSGA-II, NSGAIII, and SPEA2. The presented approach was equally compatible with other testing approaches and other evolutionary algorithms. T-wise testing and other operators such as mutation and crossover could be used, which is also possible in future work. The four objectives were proposed to test the approach by conducting experiments. As an outcome, it was figured out that the best tradeoff between the evaluated objectives was represented by the good solutions produced.

7.2. Threats to Validity

It is believed that the size of the software product line is the main challenge to our research. Yet, the software product lines that were utilized to express the proposed approach have the ability to generate productive solutions with the capability to win over the testing optimization of the feature model. The approach was assessed to concede the scalability; the purpose was to supply a large number of products that would be tactically handled by MOEA algorithms.

There exists a limitation that can be adopted in experiments conducted in the future to obtain results using different parameter settings such as population size, crossover, and mutation rate. It is always considered a difficult task to choose parameters for algorithms. To minimize the associated threats, the recommendation offered by [59] should be considered. The iteration of the experiment was 30, as there were fixed variations in the search algorithm. The iterations were performed to minimize the possibility that these results were not achieved accidentally. The evolution procedure of fitness affected the time of execution of the algorithms, where the machine hardware and running environment would help to reduce the time while scaling the implementation to a larger feature model. The implementation of the SAT solver helped to generate the configurations from an *FM* and confirm their validity for coverage.

To conclude, it is worth indicating that the aforementioned results were accomplished by the proposed approach over a reasonable number of generations (500 generations). Since thousands of executions can be adopted by search-based approaches to be effective and successful, this can be considered an achievement of the approach [41]. For future work, the authors want to extend their research using the optimization techniques mentioned in the research work of different researchers [60–69]. The concept should be examined in the future through different experiments with a considerably larger number of feature models, to compare them with those used in our proposed work. To improve the testing process in their software firms, testers would be able to examine which test suites should be selected.

Author Contributions: First Author contributed equally to the research and wrote the article. Methodology, M.K.N.; Software, M.K.N.; Validation, S.S.A. and S.M.H.; Investigation, M.K.N.; Resources, M.J.H.; Data curation, A.N. and M.A.J.; Writing—original draft, M.A.J.; Writing—review & editing, M.A.J.; Supervision, M.A.J.; Funding acquisition, S.S.A. All authors have read and agreed to the published version of the manuscript.

Funding: The authors extend their appreciation to the Deputyship for Research & Innovation, Ministry of Education in Saudi Arabia for funding this research work through the project number: IFP22UQU4320619DSR113.

Institutional Review Board Statement: Not Applicable.

Informed Consent Statement: Not Applicable.

Data Availability Statement: Data will be available upon reasonable request.

Acknowledgments: The authors extend their appreciation to the Deputyship for Research & Innovation, Ministry of Education in Saudi Arabia for funding this research work through the project number: IFP22UQU4320619DSR113.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Harman, M.; Mansouri, S.A.; Zhang, Y. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv. (CSUR)* **2012**, *45*, 1–13. [\[CrossRef\]](#)
2. Khari, M.; Kumar, P. An extensive evaluation of search-based software testing: A review. *Soft Comput.* **2019**, *23*, 1933–1946. [\[CrossRef\]](#)
3. Engström, E.; Runeson, P. Software product line testing—a systematic mapping study. *Inf. Softw. Technol.* **2011**, *53*, 2–13. [\[CrossRef\]](#)
4. Varshosaz, M.; Al-Hajjaji, M.; Thüm, T.; Runge, T.; Mousavi, M.R.; Schaefer, I. A classification of product sampling for software product lines. In Proceedings of the 22nd International Systems and Software Product Line Conference, Gothenburg, Sweden, 10–14 September 2018; Volume 1, pp. 1–13. [\[CrossRef\]](#)
5. Cawley, C.; Botterweck, G.; Healy, P.; Abid, S.B.; Thiel, S. A 3d visualisation to enhance cognition in software product line engineering. In *International Symposium on Visual Computing*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 857–868.
6. Hotz, L.; Wolter, K.; Krebs, T. *Configuration in Industrial Product Families: The ConIPF Methodology*; IOS Press: Amsterdam, The Netherlands, 2006.
7. Cawley, C.; Thiel, S.; Botterweck, G.; Nestor, D. Visualising Inter-Model Relationships in Software Product Lines. In Proceedings of the Third International Workshop on Variability Modelling of Software-Intensive Systems, Seville, Spain, 28–30 January 2009; pp. 37–44.
8. Runeson, P.; Engström, E. Regression testing in software product line engineering. In *Advances in Computers*; Elsevier: Amsterdam, The Netherlands, 2012; Volume 86, pp. 223–263.
9. Mendonca, M.; Branco, M.; Cowan, D. SPLOT: Software product lines online tools. In Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, Orlando, FL, USA, 25–29 October 2009; pp. 761–762. [\[CrossRef\]](#)
10. Pohl, K.; Böckle, G.; Van Der Linden, F. *Software Product Line Engineering: Foundations, Principles, and Techniques*; Springer: Cham Switzerland, 2005.
11. McGregor, J.D.; Northrop, L.M.; Jarrad, S.; Pohl, K. Initiating software product lines. *IEEE Softw.* **2002**, *19*, 24. [\[CrossRef\]](#)
12. Trigaux, J.C.; Heymans, P. Software product lines: State of the art. In *Product Line ENgineering of Food Traceability Software*; FUNDP-Equipe LIEL: Namur, Belgium, 2003; pp. 9–39.
13. Buchmann, T.; Schwägerl, F. Advancing negative variability in model-driven software product line engineering. In Proceedings of the Evaluation of Novel Approaches to Software Engineering: 11th International Conference, ENASE 2016, Rome, Italy, 27–28 April 2016; pp. 1–26.
14. Al-Hajjaji, M.; Lity, S.; Lachmann, R.; Thüm, T.; Schaefer, I.; Saake, G. Delta-oriented product prioritization for similarity-based product-line testing. In Proceedings of the 2017 IEEE/ACM 2nd International Workshop on Variability and Complexity in Software Design (VACE), Buenos Aires, Argentina, 27 May 2017; pp. 34–40.
15. Al-Hajjaji, M.; Thüm, T.; Lochau, M.; Meinicke, J.; Saake, G. Effective product-line testing using similarity-based product prioritization. *Softw. Syst. Model.* **2019**, *18*, 499–521. [\[CrossRef\]](#)
16. Li, X.; Wong, W.E.; Gao, R.; Hu, L.; Hosono, S. Genetic algorithm-based test generation for software product line with the integration of fault localization techniques. *Empir. Softw. Eng.* **2018**, *23*, 1–51. [\[CrossRef\]](#)
17. Krüger, J.; Al-Hajjaji, M.; Leich, T.; Saake, G. Mutation operators for feature-oriented software product lines. *Softw. Test. Verif. Reliab.* **2019**, *29*, e1676. [\[CrossRef\]](#)
18. Lity, S.; Nieke, M.; Thüm, T.; Schaefer, I. Retest test selection for product-line regression testing of variants and versions of variants. *J. Syst. Softw.* **2019**, *147*, 46–63. [\[CrossRef\]](#)
19. Guo, J.; Liang, J.H.; Shi, K.; Yang, D.; Zhang, J.; Czarnecki, K.; Ganesh, V.; Yu, H. SMTIBEA: A hybrid multi-objective optimization algorithm for configuring large constrained software product lines. *Softw. Syst. Model.* **2019**, *18*, 1447–1466. [\[CrossRef\]](#)
20. Alsewari, A.A.; Kabir, M.N.; Zamli, K.Z.; Alaofi, K.S. Software product line test list generation based on harmony search algorithm with constraints support. *Int. J. Adv. Comput. Sci. Appl.* **2019**, *10*, 605–610. [\[CrossRef\]](#)
21. Al-Hajjaji, M.; Ryssel, U.; Schulze, M. Validating Partial Configurations of Product Lines. In Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems, Leuven, Belgium, 6–8 February 2019; pp. 1–6. [\[CrossRef\]](#)
22. Wang, S.; Ali, S.; Gotlieb, A. Minimizing test suites in software product lines using weight-based genetic algorithms. In Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, Amsterdam, The Netherlands, 6–10 July 2013; pp. 1493–1500.
23. Wang, S.; Buchmann, D.; Ali, S.; Gotlieb, A.; Pradhan, D.; Liaaen, M. Multi-objective test prioritization in software product line testing: An industrial case study. In Proceedings of the 18th International Software Product Line Conference, Florence, Italy, 15–19 September 2014; Volume 1, pp. 32–41. [\[CrossRef\]](#)

24. Ensan, F.; Bagheri, E.; Gašević, D. Evolutionary search-based test generation for software product line feature models. In *Advanced Information Systems Engineering: Proceedings of the 24th International Conference, CAiSE 2012, Gdansk, Poland, 25–29 June 2012*; Proceedings 24; Springer: Berlin, Germany, 2012; pp. 613–628. [[CrossRef](#)]
25. Henard, C.; Papadakis, M.; Perrouin, G.; Klein, J.; Traon, Y.L. Multi-objective test generation for software product lines. In *Proceedings of the 17th International Software Product Line Conference*, Tokyo, Japan, 26–30 August 2013; pp. 62–71. [[CrossRef](#)]
26. Matnei Filho, R.A.; Vergilio, S.R. A mutation and multi-objective test data generation approach for feature testing of software product lines. In *Proceedings of the 2015 29th Brazilian Symposium on Software Engineering*, Belo Horizonte, Brazil, 21–26 September 2015; pp. 21–30. [[CrossRef](#)]
27. Henard, C. Enabling Testing of Large Scale Highly Configurable Systems with Search-Based Software Engineering: The Case of Model-Based Software Product Lines. Doctoral Dissertation, University of Luxembourg, Luxembourg, 2015.
28. Olaechea, R.; Rayside, D.; Guo, J.; Czarnecki, K. Comparison of exact and approximate multi-objective optimization for software product lines. In *Proceedings of the 18th International Software Product Line Conference*, Paris, France, 9–13 September 2019; Volume 1, pp. 92–101. [[CrossRef](#)]
29. Sayyad, A.S.; Menzies, T.; Ammar, H. On the value of user preferences in search-based software engineering: A case study in software product lines. In *Proceedings of the 2013 35th International Conference on Software Engineering (ICSE)*, San Francisco, CA, USA, 18–26 May 2013; pp. 492–501. [[CrossRef](#)]
30. Diaz, J.; Perez, J.; Fernandez-Sanchez, C.; Garbajosa, J. Model-to-code transformation from product-line architecture models to aspectj. In *Proceedings of the 2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, Santander, Spain, 4–6 September 2013; pp. 98–105. [[CrossRef](#)]
31. Karimpour, R.; Ruhe, G. Bi-criteria genetic search for adding new features into an existing product line. In *Proceedings of the 2013 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, San Francisco, CA, USA, 20 May 2013; pp. 34–38. [[CrossRef](#)]
32. Cruz, J.; Neto, P.S.; Britto, R.; Rabelo, R.; Ayala, W.; Soares, T.; Mota, M. Toward a hybrid approach to generate software product line portfolios. In *Proceedings of the 2013 IEEE Congress on Evolutionary Computation*, Cancun, Mexico, 20–23 June 2013; pp. 2229–2236. [[CrossRef](#)]
33. Guo, J.; White, J.; Wang, G.; Li, J.; Wang, Y. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *J. Syst. Softw.* **2011**, *84*, 2208–2221. [[CrossRef](#)]
34. Pereira, J.A.; Figueiredo, E.; Noronha, T. Modelo computacional para apoiar a configuração de produtos em linha de produtos de software. In *Proceedings of the V Workshop de Engenharia de Software Baseada em Busca (WESB). Congresso Brasileiro de Desenvolvimento de Software (CBSOFT)*, Brasilia, Brazil, 2013; pp. 80–89.
35. White, J.; Galindo, J.A.; Saxena, T.; Dougherty, B.; Benavides, D.; Schmidt, D.C. Evolving feature model configurations in software product lines. *J. Syst. Softw.* **2014**, *87*, 119–136. [[CrossRef](#)]
36. Li, J.; Liu, X.; Wang, Y.; Guo, J. Formalizing feature selection problem in software product lines using 0–1 programming. In *Practical Applications of Intelligent Systems: Proceedings of the Sixth International Conference on Intelligent Systems and Knowledge Engineering, Shanghai, China, 15–17 December 2011*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 459–465. [[CrossRef](#)]
37. Sayyad, A.S.; Goseva-Popstojanova, K.; Menzies, T.; Ammar, H. On parameter tuning in search based software engineering: A replicated empirical study. In *Proceedings of the 2013 3rd International Workshop on Replication in Empirical Software Engineering Research*, Baltimore, MD, USA, 9 October 2013; pp. 84–90. [[CrossRef](#)]
38. Sayyad, A.S.; Ingram, J.; Menzies, T.; Ammar, H. Optimum feature selection in software product lines: Let your model and values guide your search. In *Proceedings of the 2013 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, San Francisco, CA, USA, 20 May 2013; pp. 22–27. [[CrossRef](#)]
39. Lopez-Herrejon, R.E.; Chicano, F.; Ferrer, J.; Egyed, A.; Alba, E. Multi-objective optimal test suite computation for software product line pairwise testing. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, Eindhoven, The Netherlands, 22–28 September 2013; pp. 404–407. [[CrossRef](#)]
40. Henard, C.; Papadakis, M.; Perrouin, G.; Klein, J.; Heymans, P.; Le Traon, Y. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Trans. Softw. Eng.* **2014**, *40*, 650–670. [[CrossRef](#)]
41. Johansen, M.F.; Haugen, Ø.; Fleurey, F. Properties of realistic feature models make combinatorial testing of product lines feasible. In *Model Driven Engineering Languages and Systems: Proceedings of the 14th International Conference, MODELS 2011*, Wellington, New Zealand, 16–21 October 2011; Proceedings 14. Springer: Berlin/Heidelberg, Germany, 2011; pp. 638–652. [[CrossRef](#)]
42. Mendonca, M.; Wasowski, A.; Czarnecki, K. SAT-based analysis of feature models is easy. In *Proceedings of the 13th International Software Product Line Conference*, San Francisco, CA, USA, 24–28 August 2009; pp. 231–240.
43. Oster, S.; Markert, F.; Ritter, P. Automated incremental pairwise testing of software product lines. In *Software Product Lines: Going Beyond: Proceedings of the 14th International Conference, SPLC 2010, Jeju Island, Republic of Korea, 13–17 September 2010*; Proceedings 14; Springer: Berlin/Heidelberg, Germany, 2010; pp. 196–210. [[CrossRef](#)]
44. Perrouin, G.; Oster, S.; Sen, S.; Klein, J.; Baudry, B.; Le Traon, Y. Pairwise testing for software product lines: Comparison of two approaches. *Softw. Qual. J.* **2012**, *20*, 605–643. [[CrossRef](#)]

45. Al-Msie'Deen, R.; Seriai, A.D.; Huchard, M.; Urtado, C.; Vauttier, S.; Salman, H.E. An approach to recover feature models from object-oriented source code. *Actes J. Lignes Prod.* **2012**, *2012*, 15–26.
46. Jamil, M.A.; Nour, M.K.; Alhindi, A.; Awang Abhubakar, N.S.; Arif, M.; Aljabri, T.F. Towards Software Product Lines Optimization Using Evolutionary Algorithms. *Procedia Comput. Sci.* **2019**, *163*, 527–537. [[CrossRef](#)]
47. Zitzler, E.; Künzli, S. Indicator-based selection in multiobjective search. In *Parallel Problem Solving from Nature—PPSN VIII*; Springer: Berlin/Heidelberg, Germany, 2004; Volume 4, pp. 832–842. [[CrossRef](#)]
48. Zhang, Q.; Li, H. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Trans. Evol. Comput.* **2007**, *11*, 712–731. [[CrossRef](#)]
49. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T.A.M.T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [[CrossRef](#)]
50. Deb, K.; Jain, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. *IEEE Trans. Evol. Comput.* **2013**, *18*, 577–601. [[CrossRef](#)]
51. Zitzler, E.; Laumanns, M.; Thiele, L. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*; TIK Report; ETH Zurich, Computer Engineering and Networks Laboratory: Zurich, Switzerland, 2001; Volume 103. [[CrossRef](#)]
52. Hadka, D. *MOEA Framework User Guide: A Free and Open Source Java Framework for Multiobjective Optimization*; Version 2; Free Software Foundation: Boston, MA, USA, 2014; pp. 1–192.
53. Ehr Gott, M. Vilfredo Pareto and multi-objective optimization. *Doc. Math.* **2012**, *8*, 447–453.
54. García, S.; Molina, D.; Lozano, M.; Herrera, F. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 special session on real parameter optimization. *J. Heuristics* **2009**, *15*, 617–644. [[CrossRef](#)]
55. Zitzler, E.; Thiele, L.; Laumanns, M.; Fonseca, C.M.; Da Fonseca, V.G. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Trans. Evol. Comput.* **2003**, *7*, 117–132. [[CrossRef](#)]
56. Zitzler, E.; Brockhoff, D.; Thiele, L. The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration. In *Evolutionary Multi-Criterion Optimization: Proceedings of the 4th International Conference, EMO 2007, Matsushima, Japan, 5–8 March 2007*; Proceedings 4; Springer: Berlin/Heidelberg, Germany, 2007; pp. 862–876. [[CrossRef](#)]
57. Schott, J.R. *Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization*; Air Force Institute of Technology: Dayton, OH, USA, 1995; Available online: <https://hdl.handle.net/1721.1/11582> (accessed on 10 July 2023).
58. Yoo, S.; Harman, M. Pareto efficient multi-objective test case selection. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, London, UK, 9–12 July 2007; pp. 140–150. [[CrossRef](#)]
59. Arcuri, A.; Briand, L. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering*, Honolulu, HI, USA, 21–28 May 2011; pp. 1–10. [[CrossRef](#)]
60. Kader, M.A.; Zamli, K.Z.; Alkazemi, B.Y. An Experimental Study of a Fuzzy Adaptive Emperor Penguin Optimizer for Global Optimization Problem. *IEEE Access* **2022**, *10*, 116344–116374. [[CrossRef](#)]
61. Odili, J.B.; Noraziah, A.; Alkazemi, B.; Zarina, M. Stochastic process and tutorial of the African buffalo optimization. *Sci. Rep.* **2022**, *12*, 17319. [[CrossRef](#)]
62. Alsewari, A.A.; Zamli, K.Z.; Al-Kazemi, B. Generating t-way test suite in the presence of constraints. *J. Eng. Technol. (JET)* **2015**, *6*, 52–66.
63. Zamli, K.Z.; Alsewari, A.R.; Al-Kazemi, B. Comparative benchmarking of constraints t-way test generation strategy based on late acceptance hill climbing algorithm. *Int. J. Softw. Eng. Comput. Sci. (IJSECS)* **2015**, *1*, 14–26. [[CrossRef](#)]
64. Zamli, K.Z.; Mohd Hassin, M.H.; Al-Kazemi, B. tReductSA–Test Redundancy Reduction Strategy Based on Simulated Annealing. In *Intelligent Software Methodologies, Tools and Techniques: Proceedings of the 13th International Conference, SoMeT 2014, Langkawi, Malaysia, 22–24 September 2014*; Revised Selected Papers 13. Springer International Publishing: Berlin/Heidelberg, Germany, 2015; pp. 223–236. [[CrossRef](#)]
65. Wazirali, R.; Alasmay, W.; Mahmoud, M.M.; Alhindi, A. An optimized steganography hiding capacity and imperceptibly using genetic algorithms. *IEEE Access* **2019**, *7*, 133496–133508. [[CrossRef](#)]
66. Ahmad, A. Optimizing Training Data Selection for Decision Trees using Genetic Algorithms. *Int. J. Comput. Sci. Netw. Secur. (IJCSNS)* **2020**, *20*, 84.
67. Sahu, K.; Srivastava, R.K. Needs and importance of reliability prediction: An industrial perspective. *Inf. Sci. Lett.* **2020**, *9*, 33–37.
68. Sahu, K.; Srivastava, R.K. Predicting software bugs of newly and large datasets through a unified neuro-fuzzy approach: Reliability perspective. *Adv. Math. Sci. J.* **2021**, *10*, 543–555. [[CrossRef](#)]
69. Sahu, K.; Srivastava, R.K. Soft computing approach for prediction of software reliability. *Neural Netw.* **2018**, *17*, 19.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.