Pairwise Test Case Generation using (1+1) Evolutionary Algorithm for Software Product Line Testing

Sharafeldin Kabashi Khatir[1], Rabatul Aduni Binti Sulaiman[2*], Mohammed Adam Kunna Azrag[3*], Jasni Mohamad Zain[4], Julius Beneoluchi Odili[5], Samer Ali Al-Shami[6]

Faculty of Computer Science and Information Technology, University Tun Hussein Onn Malaysia, Batu Pahat, Malaysia[1]
Faculty of Computer Science and Information Technology, University Tun Hussein Onn Malaysia, Batu Pahat, Malaysia[2, 3]
Institute for Big Data Analytics & Artificial Intelligence, University Technology Mara, Shah Alam, Malaysia[2, 3]
Institute for Big Data Analytics & Artificial Intelligence, University Technology Mara, Shah Alam, Malaysia[4]
Institute of Digital Humanities, Anchor University, Lagos, Nigeria[5]
Institute of Technology Management and Entrepreneurship, University Technical Malaysia, Melaka, Malaysia[6]

***Abstract*—Software product line SPLs, or software product lines, are groups of similar software systems that share some commonalities but stand out from one another in terms of the features they offer. Over the past few decades, SPLs have been the focus of a great deal of study and implementation in both the academic and commercial sectors. Using SPLs has been shown to improve product customization and decrease time to market. Additional difficulties arise when testing SPLs because it is impractical to test all possible product permutations. The use of Combinatorial Testing in SPL testing has been the subject of extensive study in recent years. The purpose of this study is to gather and analyze data on combinatorial testing applications in SPL, apply Pairwise Testing using (1+1) evolutionary algorithms to SPL across four case studies, and assess the algorithms' efficacy using predetermined evaluation criteria. According to the findings, the performance of this technique is superior when the case study is larger, that is, when it has a higher number of features, than when the case study is smaller in scale.**

***Keywords*—SPL; SPL testing; combinatorial testing; pairwise testing; evolutionary algorithm; 1+1 EA**

I. INTRODUCTION

Software product line (SPL), which is also called software product line development, is a set of software engineering practices for making similar software systems from a single set of software assets and using the same production method for all of them [1]. In other words, SPL is a group of products that is put together based on a set of features. A Feature Model (FM) decides which products are valid. Most of the time, it's not possible to test all of the products that would come from an SPL [1]. So, a small group of these products must be chosen. It used to be best if it got a good order of products.

Effective testing strategies will help any organization that spends a lot of money on software development. This is a demand in SPL because the proportion of testing costs goes up as the costs of developing each product go down. Due to the large number of ways the base software can be changed, testing an entire product line takes a long time and costs a lot of money [2]. These issues had to do with which platform was the most efficient and what should be tested in separate products based on how hard it was to test the whole product line.

If testing is seen as a long process, the ability and effectiveness of testing can be improved by making the creation of test cases happen automatically. Even though this is a step in the right direction, more research needs to be done on the SPL testing process because it is impossible to test all of the individual systems that make up a large SPL software system. Test case generation in SPL [3], [4] is based on variation point management. The authors in [5] say that SPL testing is hard, but it would be best if all SPL products were set up correctly. In reality, though, this is hard to do. Large product configurations have made SPL testing difficult to handle, as [6]. In fact, some features can be set up in tens of millions of different ways. Since there is pressure to make test suites for the whole product line smaller, it will be hard to test each product in an SPL and stay on budget [7]. Combinatorial testing and other testing methods can cut down on the number of test suites needed for this, but they don't come without their own problems when it comes to scalability.

An Evolutionary Algorithm (EA) is a type of evolutionary computation, which is an optimization algorithm used in the field of artificial intelligence that is based on a population. In EA, processes like reproduction, mutation, recombination, and selection are used to model how natural evolution works. Even though it started in the early 1960s, EA is still a fairly new and changing field, with most research focusing on how it can be used.

The EA, as the name implies, functions similarly to natural evolution. People believe that the processes of recombination, mutation, and selection make individuals more fit because they adapt to their surroundings. An "EA individual" is a single optimization solution, whereas an "EA population" is a collection of “EA individual" optimization solutions.

Combinatorial testing is a type of testing that can be used to test a software product in a thorough way [8]. The goal is to have a product that doesn't have any bugs and can work with a wide range of inputs. Pairwise testing, also called "all-pairs

testing," is a way to check the quality of software by comparing the actual results to what was expected. Here, software testers look at all possible pairs of parameters used to test a feature and compare and contrast them.

Pairwise testing has become an important technique for any software tester over the past few years. This method has been around for almost 20 years, but it has become more popular in the last five. At least 20 tools that can make pairwise test cases have had their information made public up to this point [7, 8].

Based on the examined research, there are three main problems that sum up the issues at hand. First, there are so many possible industrial SPL products that it would be impractical to check each one individually to see if it meets each criterion. Second, it is not possible to do a full test that looks at every possible combination of parameters and values. Lastly, a way for evaluating the effectiveness of EA and creating valid comparisons, as well as a technique for reducing testing effort and shortening testing time using the suggested strategy [9]. This study's objectives are to apply the (1+1) evolutionary algorithm to generate pairwise test cases in software product line testing and to compare the effectiveness of the (1+1) EA in terms of pairwise coverage, execution time, test suite size, and test case redundancy for the mobile phone, vending machine, online shopping, and IoT device case studies.

However, it is imperative to acknowledge that the evaluation of security and privacy-related testing concerns should also be taken into account while conducting SPL testing [50]. Access control and authorization procedures are essential elements of software systems that are responsible for managing sensitive data or executing crucial operations. The process of testing these mechanisms across various products within a software product line presents challenges due to the potential variations in access control requirements and authorization procedures among different products.

In order to address this matter, it is possible to utilize sophisticated testing methodologies, such as pairwise testing, to build a set of test cases that encompass various combinations of access control criteria and authorization processes. The (1+1) evolutionary algorithm can be utilized as an efficient method for generating test cases. The efficiency of these test cases can then be assessed by utilizing metrics such as coverage and defect detection rate.

In addition to access control and authorization methods, software product line testing should also encompass additional important security and privacy-related testing concerns, including but not limited to data privacy, encryption, and secure communication. By effectively addressing these concerns [48, 49], software developers may guarantee security and ensure the privacy of their products.

Segment Particle Swarm Optimization (Se-PSO), Enhanced Segment Particle Swarm Optimization (Ese-PSO), and African Buffalo Optimization (ABO) are swarm intelligence-based optimization algorithms inspired by the behavior of social organisms such as ants, bees, and honeybees. They have been successfully applied to various optimization problems and can also be used for test case generation. These algorithms can be utilized in pairwise test case generation to generate optimal test cases that cover all possible pairwise combinations of input parameters [10, 11, 12, 13, 14, 47].

Pairwise testing is effective at reducing the number of test cases required for high coverage, but it may not be adequate for testing non-functional requirements such as performance, security, and usability. Furthermore, it has been observed in certain research that the effectiveness of pairwise testing could be reliant upon the particular attributes of the software product line under examination, including the number of features and the level of variability. Hence, additional research is required to assess the efficiency of pairwise testing across various scenarios and to develop more sophisticated methodologies for evaluating software product lines.

The intention of this study is to apply the (1+1) evolutionary algorithm to the task of generating pairwise test cases for software product line testing and to assess the efficacy of this technique using a number of measures, including pairwise coverage, execution time, test suite size, and test case redundancy. The objective is to demonstrate that the (1+1) evolutionary algorithm can be helpful for producing high-quality test cases for software product lines and to encourage the evolution of more sophisticated testing methods for software product lines.

The rest of this article is organized as follows: Section II outlines the works that are related to this study. Section III demonstrates the method for conducting this study with case studies that are used to carry out the experiment. In Section IV, a number of experiments are carried out, and the findings are thoroughly examined. Section V contains a discussion of the study's findings. Finally, in Section VI and VII, we provide a brief summary of the paper and discuss future work respectively.

II. Related Works

SPL is a collection of software-heavy systems with a common base and features tailored to a specific audience or mission. Features identify SPL members by highlighting shared and unique traits. Feature models express feature relationships and limitations to reflect all SPL outputs.

The SPL testing process is difficult. Testing every product is impractical. The number of configurations (or products) caused by an FM usually grows exponentially with the number of features, resulting in millions of potential products to test. Test engineers are trying to reduce their test suites to meet budgets and deadlines [15].

Software testing a product line takes time [16, 3]. A product line lets a buyer build a software system with many options [17]. Business is embracing SPL. Bosch, Philips, Siemens, General Motors, Hewlett-Packard, Boeing, and Toshiba use product-line approaches to reduce development and maintenance costs, improve quality, and speed product development [17, 18].

A. Software Product Line Testing

Testing software product lines is important because one bug can affect hundreds of thousands or millions of products. The study [19] lists several product line assessment methods.

Product-by-product testing begins by generating and testing concrete products one at a time using single-product testing methods. Family-based testing checks if all products in a family meet the requirement [20].

A family-based approach tests multiple products. Computer simulations represent all line products. By superimposing all product test specifications, modern family-based testing methods don't allow for good testing of software's interaction with hardware and the environment [21, 22]. Family-based testing may be time-consuming and incomplete due to its complex execution environment.

Testers and software engineers use FM to compare and create testable products. Testing all product feature combinations is not always possible. Application complexity reduces product selection. Combinatorial testing is used in the selection process to examine multiple variables. This selection method disregards FM defects. A fault-based approach like mutation-based testing can improve error detection and SPL product compliance. The research [23] suggests mutating products for SPL feature testing. The method can be used to create and evaluate test cases like a test criterion. FM's model features and connections. Feature diagrams (FDs) usually show the FM as a tree.

B. Test Case Generation Approaches in SPL

SPL test case generation has led to several testing methods. Combinatorial and model-based testing are examples. Combinatorial testing prevents tests from growing exponentially by trying all possible input permutations. Combinatorial testing addresses test selection from the whole combinatorial product since testing often has a finite test budget and exhaustive testing is usually intractable. Pairwise combinatorial testing is common here. A family of products with all FM valid pairs of features is the goal [24]. Counting covered pairs which help evaluate the product set.

All-pairs testing, also called pairwise testing, is a way to test software by giving it as many possible combinations of two inputs. This method helps us understand how inputs interact, improving product quality and dependability. Pairwise testing is useful for testing software product lines, which are collections of configurable products. Pairwise testing can improve product line testing and be applied to a case study [25]. Pairwise testing with other methods and business knowledge may reduce testing costs and improve quality [26, 37]. The paper recommends pairwise testing to reduce test cases.

T-wise testing checks all input value permutations with a constraint of "T" inputs. This strategy can help test too many inputs. T-wise testing balances test case volume and coverage [27]. SPL's model-based t-wise testing creates a TS with comprehensive t-wise coverage. A valid t-set has t features that meet some constraints.

Covering arrays in software product line testing improves system failure detection [28]. For testing, a two-layer covering array is used to represent equivalence classes and compute their names in the second layer. Covering arrays are used to test component interactions in a systematic manner. Let N, t, k, and v be integers with $k \geqslant t \geqslant 2$ and $v \geqslant 2$. A covering array CA (N; t, k, v) is an $N \times k$ array A in which each entry is from a different alphabet, and there is a row of B that equals x for every $N \times t$ subarray B of A and every $x \in \Sigma \wedge t$. Then t denotes the covering array's strength, k the number of factors, and v the number of levels [29].

Model-based testing (MBT) automates test case creation for SPL testing. A Systematic Literature Review (SLR) on MBT for SPL testing is presented by [30]. MBT in SPL issues, evaluation, and solutions are discussed. The study summarizes SPL MBT perspectives in a taxonomic structure. The latest SPL development is taxonomy based MBT classification.

Reduced testing is needed when resources are limited. Risk-based testing [31] is popular for system prioritization. Two other factors determine the probability of system entity damage or loss.

SPL regression testing is difficult because it must test every member of a product family after a change. Regression test selection (RTS) selects a subset of regression test cases to lower regression testing costs [32]. In the product line context, each test case can be executed on multiple products that reuse the test case, making SPL regression testing time-consuming and resource-intensive even with RTS. Eliminating unnecessary test case executions helps.

In [33], a suggested method that finds a group of products where running the test case will cover the same sequence of source code statements and give the same testing results, and then filters out the group from the test case's scope.

C. Search-based Techniques for SPL Testing

SPLs are collections of systems that have the same core functionality but are tailored to meet the needs of specific user groups. All products would have to be tested in theory, but that's not possible in practice. Because of this, "interesting" ones can be chosen to focus on using search-based approaches.

Evolutionary computation is a population-based metaheuristic optimization algorithm used in artificial intelligence research. EA simulates natural evolution using reproduction, mutation, recombination, and selection. EAs mimic natural evolution. Recombination, mutation, and selection are thought to increase fitness by adapting individuals to their environment. EA "population" members are optimization solutions. EAs excel at optimization, scheduling, planning, design, and management [34]. Investments, production, distribution, etc. have these issues.

Initially, a theoretical study of the (1+1) EA is presented and discussed. On a population of one, it only employs the mutation operation and an elitist selection method to generate a new generation. Although the (1+1) EA is the simplest evolutionary algorithm, it shares a fundamental principle with all others [35]. The (1+1) EA locates the maximum of a linear function, as proven by a theorem in [28]. The two members of the population at any given iteration are known as the "parent" and the "offspring," hence the name "1+1." For linear function optimization, the (1+1) Evolutionary Algorithm is predicted to take O (n ln n) time if the mutation rate is of size (1/n).

Differential evolution (DE), Evolution strategy (ES), and Evolutionary Programming (EP) are all examples of other

Evolutionary Algorithms that can produce multiple offspring and compete [36]. As an illustration, the Evolution Strategy allows for the creation and competition of mutants.

Table I provides a summary of some of the existing search-based techniques for testing in SPL. Moreover, the strengths and weaknesses of the technique are provided as well.

TABLE I. SUMMARY OF STUDIED SEARCH-BASED TESTING TECHNIQUE

Technique	Authors	Strengths	Weaknesses
1+1 Evolutionary Algorithm (1+1 EA)	Slowik & Kwasnicka, Zhou et al 2020. [36]	Simplest EA, requires low requirements and it can reach any point in the search space in a single step.	it's not easy to find a good drift function.
Genetic Algorithm (GA)	Rao & Tripathy 2019. [38]	The ability to make exceptional use of parallel computation, simplicity of use, rapid convergence to the global optimum, few necessary control variables.	Do not scale well with complexity, can be quite slow.
Non-dominated Sorting Genetic Algorithm II (NSGA-II)	Hojjati et al., Muhammad Abid Jamil et al 2018. [39]	Demonstrates elitism and is not dependent on any measure of distributivity.	The computational complexity of solving the problem grows in proportion to the size of the problem.
Strength Pareto Evolutionary Algorithm II (SPEA-II)	Jamil et al 2019. [31]	Utilizes a fine-grained fitness assignment strategy and an improved archive truncation technique.	lack of accuracy in its density estimation

III. METHODOLOGY

The methodology for conducting the research includes four stages. The first thing that will be done is an analysis of the software product line online tools (SPLOT), and then in Step 1, the FM for all of the case studies will be prepared. Following that, the pairwise testing will be carried out using the (1+1) EA in Step 2. Evaluation of the parameters that were employed is the third step. Lastly, an in-depth analysis and comparison of the results is carried out. The research methodology is depicted in Fig. 1.

A. Step 1: Prepare Case Studies using Software Product Lines Online Tools (SPLOT)

SPLOT is a Java2 Web app that uses an HTML template engine to make Ajax-based user interfaces for reasoning and configuration. Because it is web-based, you don't have to update it by hand or download any files, and it's easy to share information (for example, through a feature model repository). Automated reasoning and product configuration are SPLOT's two main offerings right now. To this end, reasoning is centered on the automation of crucial debugging tasks like checking the consistency of feature models and spotting the presence of dead and common features [6]. Measurements of properties like the number of valid configurations and the degree of variability of feature models are also supported by reasoning. Currently, SPLOT supports interactive configuration for product configuration, wherein users decide at a time, and the configuration system automatically propagates those decisions to enforce their consistency.

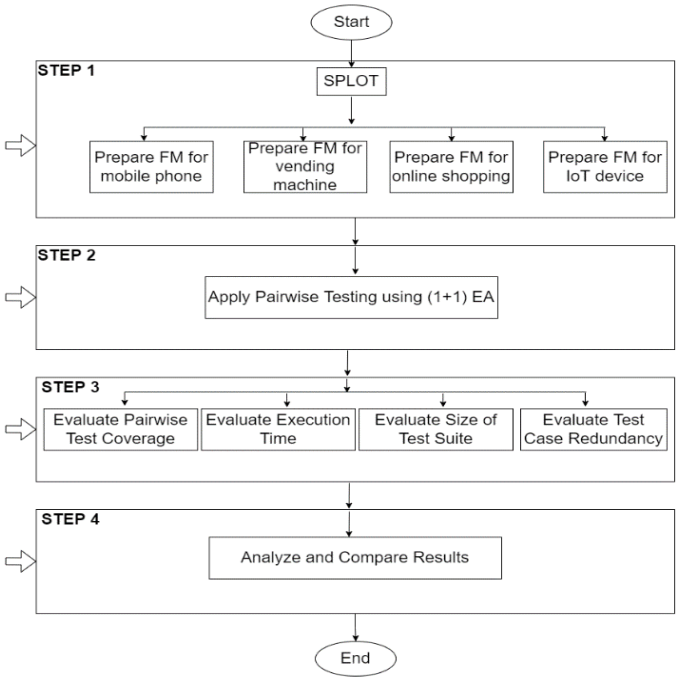

Fig. 1. Research methodology framework.

A major complaint from SPL researchers is the dearth of freely distributed feature models. To address this problem, SPLOT makes available a public model repository with more than 20 genuine models from the literature as well as several automatically generated models with up to 10,000 features each [6].

B. Step 2: Apply Pairwise Testing using (1+1) EA

In this step, we use the PLEDGE tool to generate (1+1)-based pairwise test cases. PLEDGE is a free software program that helps determine which product configurations should be tested in order to cover the most possible combinations of features. Both a command line and a graphical user interface are available for this tool's operation (GUI). All of the following are possible with the current release of PLEDGE:

- FMs loaded from a file: Both the SPLOT and DIMACS (Conjunctive Normal Form) formats are supported by PLEDGE [32].
- Information about FM, such as its limitations and characteristics, can be visualized.
- Making changes to the FM by introducing or removing constraints.
- Producing the test products from the FM by setting parameters for the desired quantity and the time allotted for their production.

- Inputting a list of products and sorting them into a desired order using one of two suggested methods of prioritization.
- Producing a file to store the finalized or prioritized output.

Several settings can be specified to modify the behavior of the 1+1 evolutionary algorithm when using PLEDGE to conduct pairwise testing using the 1+1 evolutionary algorithm.

C. Step 3: Evaluate Testing Parameters

In this stage, all of the parameters, such as pairwise coverage, execution time, the size of test suites, and test case redundancy are evaluated based on the methodology that was utilized in this study.

1) Pairwise coverage: For each case study in this research is given a percentage of pairwise coverage for a single run/iteration, which is the discrete combinations of the relevant parameters calculated with PLEDGE. When testing in a black box, pairwise coverage ensures that every possible combination of input parameters is covered by at least one of the test cases. Pairwise testing is more efficient at spotting problems because it is based on the observation that most defects occur due to the interaction of two values. By allowing for systematic testing coverage, pairwise tools can speed up the preparation and implementation stages. By conducting tests in pairs, we can reduce testing time by half without compromising coverage.

The use of k-means and k-medoids clustering techniques in software testing to reduce the test suite and improve the algorithm's performance is discussed by [40, 41]. The technique of pairwise testing is also cited as an efficient means of generating a small test suite with optimal pairwise coverage. Also, [42] proposes a new method for increasing testing efficiency while maintaining testing efficacy. The paper ranks combinatorial test cases according to incremental interaction coverage by repeatedly applying the base choice coverage.

2) Execution time: The execution time is the amount of time it takes for a single run to be carried out. Also using PLEDGE, the execution time in seconds is available. For each run, the time taken to finish the run is provided. In the context of pairwise testing, execution time refers to the amount of time required to execute the test cases created using the all-pairs or pairwise testing methodology. The execution time is dependent on variables such as the size of the input parameters, the number of combinations, the performance of the being tested software application, and the testing environment.

Reducing test execution time is crucial for SPL testing, as it enables more efficient testing of product lines and reduces the need for unnecessary testing [43]. Several SPL testing methods have been proposed to decrease test execution redundancy and boost efficiency.

3) Size of test suite: A test suite consists of all the test cases that have been logically grouped together. Testing an application to show that it exhibits a certain set of behaviors is what the test suite is all about. Each test case in a suite will have explicit instructions or goals and details on the system configuration to be used during testing. [41] emphasizes the significance of pairwise testing as a means to circumvent the combinatorial explosion issue. The paper proposes that pairwise testing can be used to test software systems' vast input combinations with fewer test cases. Pairwise testing is presented in [28] as a promising technique with the potential to drastically reduce the number of test cases required for an acceptable level of coverage.

For each case study involved in this study, a different size of test suite can be generated for a single run using PLEDGE. The size of test suites can differ based on the number of features for each case study, a case study with many features is considered big and size of test suites can be high.

4) Test case redundancy: This study examines a test suite by finding redundant test cases, which is essential for lowering testing costs. A redundancy score is defined by the redundancy formula, which determines the score by dividing the total number of test cases by the number of duplicates. The redundancy score can be calculated using the formula in Equation 1 below, the total number of redundant test cases is divided by the total number of test cases generated. [44-45] contends that redundancy in test artefacts reduces testing costs.

$$Redundancy\ Score = \frac{\sum Redundant\ test\ cases}{\sum Test\ cases} \quad (1)$$

D. Step 4: Analyze and Compare the Results

The fourth step is to perform an analysis and comparison of the results, which includes testing and an evaluation of performance. The results of the testing will be analyzed and compared using pairwise coverage, execution time, total test suite size as the criteria. In addition to this, test case redundancy will be calculated, and a graph depicting the findings of the comparison will be offered.

E. Case Studies

Within the scope of this research, four distinct case studies will each be subjected to a pairwise test case generation technique utilizing (1+1) EA. The mobile phone, the vending machine, the online shopping, and the IoT device are the case studies. The reason for choosing the selected case studies is because they are the most used and because they meet the needs to conduct this research, besides, there are many references that has been used those case studies to conduct testing in SPL using other testing techniques. Mobile phone and vending machine case studies are the small case studies in term of number of features. Meanwhile, e-shop and IoT device case studies are the big case studies.

1) Mobile phone: The mobile phone industry served as inspiration for the simplified feature model shown in Fig. 2. This model demonstrates how features are incorporated into the process of specifying and developing software for mobile devices, specifically mobile phones. The capabilities of the phone will determine the types of software that can be installed on it. The model stipulates that all mobile devices

must be capable of making and receiving calls as well as displaying data in black-and-white, color, or at a very high resolution on their screens. In addition, the software for mobile phones may, at the user's discretion, include support for satellite navigation systems (GPS) and multimedia devices, such as cameras, MP3 players, or both.

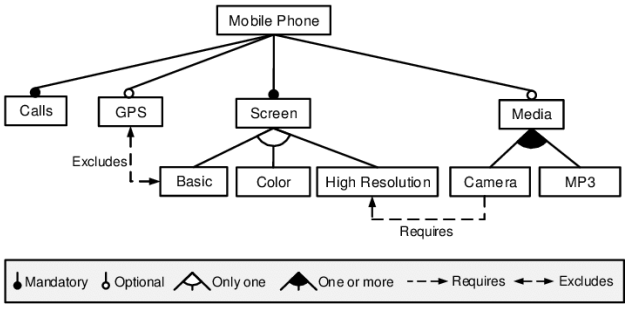

Fig. 2. Feature diagram of product line mobile phone adapted from [38].

2) Vending machine: The FD for the snack and drink dispensing machine's SPL is shown in Fig. 3. The vending machine assortment here is formally described by the accompanying feature diagram. Soda, Tea, Free Drinks, and CancelPurchase are used in the feature diagram to represent these products as valid options for the consumer.

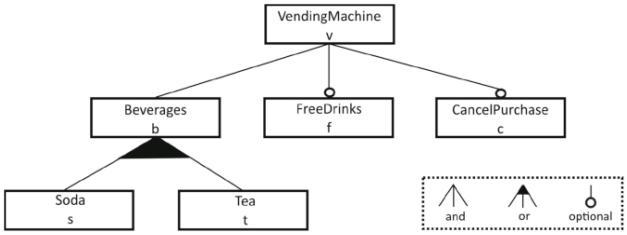

Fig. 3. Feature diagram of product line vending machine adapted from [39].

3) Online shop: The feature model that contains information about our online shops is depicted in Fig. 4. The name of the product line is located within the most prominent feature. There are four aspects that are connected to it: The features Catalog, Payment, and Security are connected to the feature that is at the top of the list by arcs that have filled circles at their ends. This indicates that these three features are required, meaning that they are present in each and every product variation. The fact that the Search function is not required is indicated by an arc that terminates in a circle that is not filled in. This descending order of characteristics will continue. For example, the feature Payment includes three sub features: Bank Account, ECoins, and Credit Card. For each product variant, at least one of these sub features must be selected. Both the High and Low sub features of the Security feature are alternative features, which means that only one of them can be selected for each product variant. In addition, there is a textual condition that states that selecting credit cards is only possible when the security level that is being provided is of a high standard.

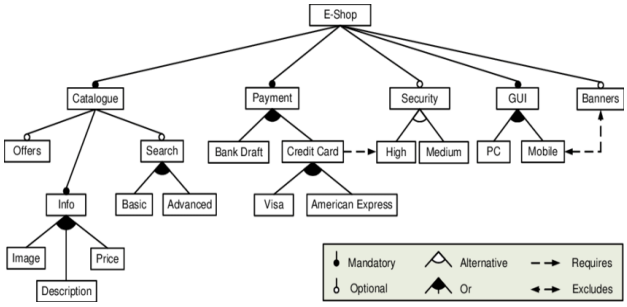

Fig. 4. Feature diagram of product line online shop adapted from [40].

4) IoT device: Internet of Things application development is guided by the selection of relevant environmental features and the needs of the end user. An efficient modelling approach, capable of holding all constraints and allowing application development, can be used to control environmental variability. Different uses for the same IoT devices introduce contextual variations that must be managed to ensure efficient development and maximize code reuse. It has been suggested that XML-based feature modelling be used to handle variability management of SPL. Fig. 5 depicts the smart campus IoT system's feature model, complete with predefined relationships and constraints.

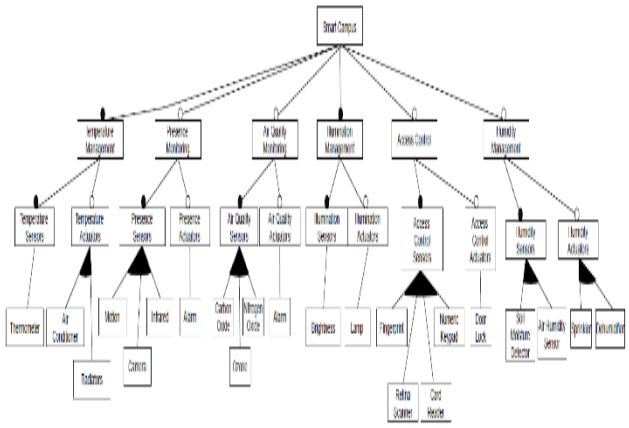

Fig. 5. Smart campus IoT feature diagram adapted from [46].

IV. Results

In this research, four case studies have been tested using the proposed approach. Each case study is tested ten times, and the results of each test case are provided. The four test cases, namely mobile phone, vending machine, online shop, and IoT, are different from each other's, in terms of the number of features, which indicates their size.

Fig. 6 shows the average results of each case study based on the evaluation metrics. For the mobile phone case study, the average test case coverage is 85.23%, and the average execution time is 1.31 seconds. The average number of test cases generated is 2.4, and no test cases are redundant. On the other hand, for the vending machine case study, the averages for the test coverage, execution time, size of the test suite, and percentage of test case redundancy are 75.39%, 1.13 seconds, 2.4, and 23.33%, respectively.

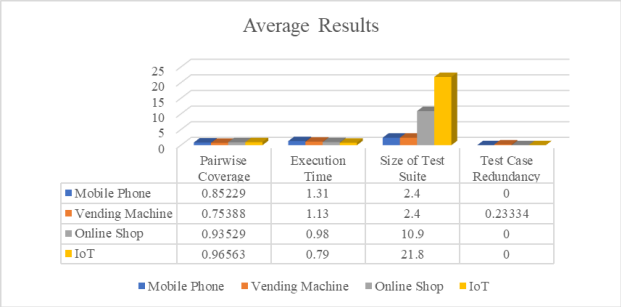

Fig. 6. Average of the results of the case studies.

Moreover, the online shop and smart campus IoT case studies are the big ones in terms of the number of features in this study, and the average percentage of test case coverage is 93.53% for the online shop and 96.56% for the smart campus. Both case study results show the absence of redundant test cases. Meanwhile, the average execution time is 0.98 and 0.79 for the online shop and IoT smart campus, respectively. Lastly, the average size of test cases is 10.9 for the online shop case study and 21.8 for the smart campus case study.

Using PLEDGE to conduct the testing, as shown previously in the mobile phone and vending machine case studies, produced less efficient results compared to the other two case studies. The size of the test suite is smaller because there are fewer features; meanwhile, there are more redundant test cases because, using PLEDGE, the results showed that the tool cannot produce a large number of test suites for small case studies, and the results also showed that the testing takes more time to run, which is remarkably unexpected.

V. Discussion

This study's goals are to apply the (1+1) evolutionary algorithm to generate pairwise test cases in software product line testing with the help of the proposed tool. The effectiveness of the algorithm using four metrics, including pairwise coverage, execution time, size of the test suite, and test case redundancy, for all of the case studies that were chosen, and to conclude that the (1+1) evolutionary algorithm is useful for generating pairwise test cases in software product line testing.

The literature research clarifies the difficulties associated with testing software product lines, mostly stemming from the wide range of possible configurations and the need for effective testing methodologies. The findings of this study indicate that the (1+1) evolutionary algorithm is a successful approach for generating pairs test cases in the context of software product line testing. This algorithm proves to be effective in reducing the number of required test cases while simultaneously obtaining a high level of coverage.

Furthermore, the literature review examines the significance of assessing the efficiency of testing methodologies by considering several factors, including pairwise coverage, execution time, and test suite size. The study presents a comparative analysis of the effectiveness of the (1+1) evolutionary algorithm across four distinct case studies, shedding light on its performance in diverse circumstances.

Moreover, the literature review underscores the necessity for more research and advancement in the domain of software product line testing, encompassing the utilization of search-based algorithms like evolutionary algorithms. The findings and methodology presented in the study make a valuable contribution to the field of research by illustrating the successful performance of the (1+1) evolutionary algorithm in producing pairs test cases for software product line testing.

VI. Conclusion

The purpose of this research is to apply the (1+1) evolutionary algorithm using PLEDGE in order to generate pairs of test cases for software product line testing. Using four criteria—pairwise coverage, execution time, test suite size, and test case redundancy—for each of the selected case studies, it was discovered that the 1+1 evolutionary algorithm is beneficial for creating pairwise test cases in software product line testing. When using PLEDGE to conduct the testing, the results demonstrated that this method yields superior results when the case study is large, which means it has a large number of features, compared to when the case study is small.

Among the four case studies, the online shop and IoT case studies achieved good results in comparison to the mobile phone and vending machine case studies. This is because the online shop and IoT case studies have a large number of features; therefore, by using the PLEDGE tool, a good result has been achieved in comparison to when a case study has a small number of features, such as the mobile phone and vending machine case studies.

Online shop and IoT case studies achieved better results than mobile phone and vending machine case studies. The average for pairwise coverage recorded the best for online shop and IoT at 93.53% and 96.56%, respectively. Meanwhile, the average execution time and the size of test suites are noted to be better for online shop and IoT case studies at 0.98s and 0.79s, 10.9s and 21.8s respectively. Also, both case studies showed the absence of redundant test cases. On the other hand, the finding revealed that the mobile phone and vending machine case studies achieved less performance due to the fact that both were considered small with a small number of features.

VII. Future Works

From this research, we were able to gather the following list of significant insights regarding areas for possible development and improvement: first, there are numerous ways to produce test cases using the existing software testing tools. Second, the program used in this research, PLEDGE, has problems and must be executed multiple times before producing meaningful results. The production and prioritization of test cases is a crucial aspect of SPL testing, and there has been an increasing trend in recent years to leverage search-based algorithms as a solution strategy.

Acknowledgment

This research was supported by University Tun Hussein Onn Malaysia (UTHM) through Tier 1 Grant (Vot H937).

REFERENCES

[1] Hierons, R. M., Li, M., Liu, X., Parejo, J. A., Segura, S., & Yao, X. (2020). Many-objective test suite generation for software product lines. ACM Transactions on Software Engineering and Methodology, 29(1). https://doi.org/10.1145/3361146.

[2] Cico, O., Jaccheri, L., Nguyen-Duc, A., & Zhang, H. (2021). Exploring the intersection between software industry and Software Engineering education-A systematic mapping of Software Engineering Trends. Journal of Systems and Software, 172, 110736.

[3] Souza, M. R. D. A., Veado, L., Moreira, R. T., Figueiredo, E., & Costa, H. (2018). A systematic mapping study on game-related methods for software engineering education. Information and software technology, 95, 201-218.

[4] Lee, J., Kang, S., & Jung, P. (2020). Test coverage criteria for software product line testing: Systematic literature review. Information and Software Technology, 122, 106272.

[5] Santos, I., Melo, S. M., de Souza, P. S. L., & Souza, S. R. (2019, September). Testing techniques selection: A systematic mapping study. In Proceedings of the XXXIII Brazilian Symposium on Software Engineering (pp. 347-356).

[6] Horcas, J. M., Pinto, M., & Fuentes, L. (2019, September). Software product line engineering: a practical experience. In Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A (pp. 164-176).

[7] Wang, R., Artho, C., Kristensen, L. M., & Stolz, V. (2020, December). Multi-objective Search for Model-based Testing. In 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS) (pp. 130-141). IEEE.

[8] Al-Hajjaji, M., Thüm, T., Lochau, M., Meinicke, J., & Saake, G. (2019a). Effective product-line testing using similarity-based product prioritization. Software & Systems Modeling, 18, 499-521.

[9] Dominka, S., Mandl, M., Dubner, M., & Ertl, D. (2018). Using combinatorial testing for distributed automotive features: Applying combinatorial testing for automated feature-interaction-testing. 2018 IEEE 8th Annual Computing and Communication Workshop and Conference, CCWC 2018, 2018-January, 490–495. https://doi.org/10.1109/CCWC.2018.8301632.

[10] Kunna, Mohammed Adam, Tuty Asmawaty Abdul Kadir, Muhammad Akmal Remli, Noorlin Mohd Ali, Kohbalan Moorthy, and Noryanti Muhammad. "An enhanced segment particle swarm optimization algorithm for kinetic parameters estimation of the main metabolic model of Escherichia coli." Processes 8, no. 8 (2020): 963.

[11] Azrag, Mohammed Adam Kunna, Tuty Asmawaty Abdul Kadir, and Aqeel S. Jaber. "Segment particle swarm optimization adoption for large-scale kinetic parameter identification of Escherichia Coli metabolic network model." IEEE Access 6 (2018): 78622-78639.

[12] Azrag, Mohammed Adam Kunna, Jasni Mohamad Zain, Tuty Asmawaty Abdul Kadir, Marina Yusoff, Aqeel Sakhy Jaber, Hybat Salih Mohamed Abdlrhman, Yasmeen Hafiz Zaki Ahmed, and Mohamed Saad Bala Husain. "Estimation of Small-Scale Kinetic Parameters of Escherichia coli (E. coli) Model by Enhanced Segment Particle Swarm Optimization Algorithm ESe-PSO." Processes 11, no. 1 (2023): 126.

[13] Azrag, Mohammed Adam Kunna, Tuty Asmawaty Abdul Kadir, and Noorlin Mohd Ali. "A Comparison of Particle Swarm optimization and Global African Buffalo Optimization." In IOP Conference Series: Materials Science and Engineering, vol. 769, no. 1, p. 012034. IOP Publishing, 2020.

[14] Odili, Julius Beneoluchi, Mohd Nizam Mohmad Kahar, Shahid Anwar, and Mohammed Adam Kunna Azrag. "A comparative study of African buffalo optimization and randomized insertion algorithm for asymmetric travelling salesman's problem." In 2015 4th International Conference on Software Engineering and Computer Systems (ICSECS), pp. 90-95. IEEE, 2015.

[15] Onipede, S. F., Bashir, N. A., & Abubakar, J. (2022). Small open economies and external shocks: an application of Bayesian global vector autoregression model. Quality & Quantity. https://doi.org/10.1007/s11135-022-01423-8.

[16] Zhang, Y., Kong, W., Li, D., & Liu, X. (2020, October). Design and Implementation of Automatic Matching and Remote Screening System for Intelligent Security Inspection. In Proceedings of the 2020 International Conference on Computers, Information Processing and Advanced Education (pp. 76-83).

[17] Edded, S., Sassi, S. B., Mazo, R., Salinesi, C., & Ghezala, H. B. (2019). Collaborative configuration approaches in software product lines engineering: A systematic mapping study. Journal of Systems and Software, 158, 110422.

[18] Ruland, S., Lochau, M., & Jakobs, M. C. (2020). HybridTiger: Hybrid model checking and domination-based partitioning for efficient multi-goal test-suite generation (competition contribution). Fundamental Approaches to Software Engineering, 12076, 520.

[19] Kolesnikov, S., Siegmund, N., Kästner, C., Grebhahn, A., & Apel, S. (2019). Tradeoffs in modeling performance of highly configurable software systems. Software & Systems Modeling, 18, 2265-2283.

[20] Mesa, O., Vieira, R., Viana, M., Durelli, V. H., Cirilo, E., Kalinowski, M., & Lucena, C. (2018, September). Understanding vulnerabilities in plugin-based web systems: an exploratory study of wordpress. In Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1 (pp. 149-159).

[21] Lee, J., Kang, S., & Jung, P. (2020). Test coverage criteria for software product line testing: Systematic literature review. Information and Software Technology, 122, 106272.

[22] Sulaiman, R. A., Jawawi, D. N., & Halim, S. A. (2023). Cost-effective test case generation with the hyper-heuristic for software product line testing. Advances in Engineering Software, 175, 103335.

[23] Al-Hajjaji, M., Thüm, T., Lochau, M., Meinicke, J., & Saake, G. (2019). Effective product-line testing using similarity-based product prioritization. Software and Systems Modeling, 18(1), 499–521. https://doi.org/10.1007/s10270-016-0569-2

[24] Akimoto, H., Isogami, Y., Kitamura, T., Noda, N., & Kishi, T. (2019, December). A prioritization method for spl pairwise testing based on user profiles. In 2019 26th Asia-Pacific Software Engineering Conference (APSEC) (pp. 118-125). IEEE.

[25] Wang, Y., Sun, Y., Wu, X., Shanghai cai jing da xue, Tong ji da xue (China), Suzhou da xue, Institute of Electrical and Electronics Engineers. Beijing Section, & Institute of Electrical and Electronics Engineers. (2018). Proceedings of the 2018 IEEE International Conference on Progress in Informatics and Computing : December 14-16, 2018, Suzhou, China.

[26] Morgan, J. (2018). Combinatorial testing: an approach to systems and software testing based on covering arrays. Analytic methods in systems and software testing, 131-158.

[27] Xiang, Y., Huang, H., Member, S., Li, M., Li, S., & Yang, X. (2020). Looking For Novelty in Search-based Software Product Line Testing. In IEEE TRANSACTIONS ON SOFTWARE ENGINEERING.

[28] Rabatul Aduni Sulaiman, Dayang Norhayati Abang Jawawi, & Shahliza Abdul Halim. (2022). Classification Trends Taxonomy of Model-based Testing for Software Product Line: A Systematic Literature Review. KSII Transactions on Internet and Information Systems, 16(5). https://doi.org/10.3837/tiis.2022.05.008

[29] Jahan, H., Feng, Z., & Mahmud, S. H. (2020). Risk-based test case prioritization by correlating system methods and their associated risks. Arabian Journal for Science and Engineering, 45, 6125-6138.

[30] Jung, P., Kang, S., & Lee, J. (2020). Efficient regression testing of software product lines by reducing redundant test executions. Applied Sciences (Switzerland), 10(23), 1–21. https://doi.org/10.3390/app10238686

[31] Slowik, A., & Kwasnicka, H. (2020a). Evolutionary algorithms and their applications to engineering problems. In Neural Computing and Applications (Vol. 32, Issue 16, pp. 12363–12379). Springer. https://doi.org/10.1007/s00521-020-04832-8

[32] Huang, Z., Zhou, Y., Xia, X., & Lai, X. (2020). An improved (1+1) evolutionary algorithm for k-median clustering problem with performance guarantee. Physica A: Statistical Mechanics and Its Applications, 539. https://doi.org/10.1016/j.physa.2019.122992

[33] Slowik, A., & Kwasnicka, H. (2020). Evolutionary algorithms and their applications to engineering problems. Neural Computing and Applications, 32, 12363-12379.

[34] Rao, D. S., & Tripathy, D. P. (2019). A genetic algorithm approach for optimization of machinery noise calculations. Noise and Vibration Worldwide, 50(4), 112–123. https://doi.org/10.1177/0957456519839409.

[35] Hojjati, A., Monadi, M., Faridhosseini, A., & Mohammadi, M. (2018). Application and comparison of NSGA-II and MOPSO in multi-objective optimization of water resources systems. Journal of Hydrology and Hydromechanics, 66(3), 323–329. https://doi.org/10.2478/johh-2018-0006.

[36] Jamil, M. A., Nour, M. K., Alhindi, A., Awang Abhubakar, N. S., Arif, M., & Aljabri, T. F. (2019). Towards Software Product Lines Optimization Using Evolutionary Algorithms. Procedia Computer Science, 163, 527–537. https://doi.org/10.1016/j.procs.2019.12.135.

[37] Hierons, R. M., Li, M., Liu, X., Parejo, J. A., Segura, S., & Yao, X. (2020). Many-objective test suite generation for software product lines. ACM Transactions on Software Engineering and Methodology (TOSEM), 29(1), 1-46.

[38] Huang, S., Sun, J., & Feng, Y. (2018). Pairwise covariates-adjusted block model for community detection. arXiv preprint arXiv:1807.03469.

[39] Al-Hajjaji, M., Thüm, T., Lochau, M., Meinicke, J., & Saake, G. (2019b). Effective product-line testing using similarity-based product prioritization. Software & Systems Modeling, 18, 499-521.

[40] Di Silvestro, F. (2020). Improving testing reusability and automation for software product lines.

[41] Din, F., & Zamli, K. Z. (2019). Pairwise Test Suite Generation Using Adaptive Teaching Learning-Based Optimization Algorithm with Remedial Operator (pp. 187–195). https://doi.org/10.1007/978-3-319-99007-1_18

[42] Jung, P., Kang, S., & Lee, J. (2020). Efficient regression testing of software product lines by reducing redundant test executions. Applied Sciences, 10(23), 8686.

[43] Ngoumou, A., & Ndjodo, M. F. (2018). Feature-Relationship Models: A Paradigm for Cross-hierarchy Business Constraints in SPL. International Journal of Computer Science and Information Security (IJCSIS), 16(9).

[44] Dubslaff, C. (2019). Compositional feature-oriented systems. In Software Engineering and Formal Methods: 17th International Conference, SEFM 2019, Oslo, Norway, September 18–20, 2019, Proceedings 17 (pp. 162-180). Springer International Publishing.

[45] Sulaiman, R. A. B. (2020). Cost-Effective Model-Based Test Case Generation and Prioritization For Software Product Line (Doctoral dissertation, Universiti Teknologi Malaysia).

[46] Corradini, F., Fedeli, A., Fornari, F., Polini, A., & Re, B. (2021). FloWare: An Approach for IoT Support and Application Development. Lecture Notes in Business Information Processing, 421, 350–365. https://doi.org/10.1007/978-3-030-79186-5_23.

[47] Wang, Z. J., Yang, Q., Zhang, Y. H., Chen, S. H., & Wang, Y. G. (2023). Superiority combination learning distributed particle swarm optimization for large-scale optimization. Applied Soft Computing, 136, 110101. https://doi.org/10.1016/J.ASOC.2023.110101.

[48] Tauqeer, O. B., Jan, S., Khadidos, A. O., Khadidos, A. O., Khan, F. Q., & Khattak, S. (2021). Analysis of security testing techniques. Intelligent Automation & Soft Computing, 29(1), 291-306.

[49] Campanile, L., Iacono, M., & Mastroianni, M. (2022, September). Towards privacy-aware software design in small and medium enterprises. In 2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech) (pp. 1-8). IEEE.

[50] Stanciu, A. M. (2023). Theoretical Study of Security for a Software Product. In Intelligent Sustainable Systems: Selected Papers of WorldS4 2022, Volume 1 (pp. 233-242). Singapore: Springer Nature Singapore.