A Comparison on Similarity Distances and Prioritization Techniques for Early Fault Detection Rate

Safwan Abd Razak, Mohd Adham Isa and Dayang Norhayati Abang Jawawi
Faculty of Computing, Universiti Teknologi Malaysia, 81310 Johor Bahru, Johor, Malaysia.
safwan7@live.utm.my

***Abstract*—Nowadays, the Software Product Line (SPL) had replaced the conventional product development system. Many researches have been carried out to ensure the SPL usage prune the benefits toward the recent technologies. However, there are still some problems exist within the concept itself, such as variability and commonality. Due to its variability, exhaustive testing is not possible. Various solutions have been proposed to lessen this problem. One of them is prioritization technique, in which it is used to arrange back the test cases to achieve a specific performance goal. In this paper, the early fault detection is selected as the performance goal. Similarity function is used within our prioritization approach. Five different types of prioritization techniques are used in the experiment. The experiment results indicate that the greed-aided-clustering ordered sequence (GOS) shows the highest rate of early fault detection.**

***Index Terms*—Product-Line Testing; Prioritization; Software Product Lines.**

I. INTRODUCTION

Software Product Line (SPL) is a group of software-intensive systems that sharing an identical, managed group of features that fulfill the needs of a certain market section or goal and are build up from a familiar set of core assets in a recommended way [1]. SPL can give many benefits toward various organizations due to its implementation of business and technical strategy. Such benefit in software development is that SPL approach can make enhancements in time to market, cost, and reliability. This benefit not only helps the organizational, but also individual SPL practitioner [1]. Thus, numerous software organizations alter their development of software from single systems to SPLs [2].

In achieving these benefits, a complete set of activities that validate and verify the correctness of the product built should be defined. Thus, the testing approach is introduced. The product line testing is about extracting a set of products and testing each of it [3]. Testing an SPL is a hard task. This is because of the combinatorial explosion faced due to a great number of possible combination features. Exhaustive testing is infeasible. Exhaustive testing is a test approach in which all possible data combinations are used for testing. Time consuming and cost issues arise when exhaustive testing in SPLs is conducted. Many attempts have been done to solve the issues. One of them is the test case prioritization.

Test case prioritization techniques arrange test cases for execution in an order that achieves to increase their effectiveness at meeting certain performance goals [4][5]. Various goals can be specified. For examples, the software testers may want to order their test cases in an order that can attain full code coverage as soon as possible or in an order that can increase the rate of fault detection. State a goal first, then several ordering criteria can be considered. For an example, given the goal is to increase the rate of fault detection of the test cases. Software testers could order the test cases according to the presumed dispose error of the component under test or they also could order the test cases according to the number of faults detected by the previous executed test cases.

This paper presents some test case prioritization approaches for the SPLs. We explore the applicability of the similarity distance with the prioritization technique to increase the rate of early fault detection. Four type of similarity functions are used. These functions are Hamming distance, Jaccard distance, Counting function, and Sorensen-Dice. The reason we used prioritization based on similarity function is that it has higher feature coverage and higher fault detection rate [6]. Each of these similarity functions then are prioritized with five different prioritization techniques.

For the evaluation, we used the set of configurations and fault metric provided by Al-Hajjaji et al. [7]. Fault metric is the distribution of fault found in each configuration. Configuration is a valid combination of features. For each of the configuration, we calculated the similarity distances between the configurations. The result will be a table of distances between each configuration. Four tables of four similarity functions are obtained. The distances obtained are used to prioritize the configurations. Five prioritization techniques are used to prioritize each of the four tables produced. Finally, after the prioritization process is complete, we calculate the average percentage of faults detected.

The rest of the paper are organized as follows. Section II contain the related works. Section III is about the similarity distances used for the evaluation. Section IV is for the prioritization techniques used for the evaluation. The evaluation of the approaches is described in section V. Lastly, the conclusions and future work plan in Section VI.

II. RELATED WORKS

Similarity function is introduced to maximize the diversity of configurations. On the other hand, test case prioritization technique schedules the configurations for execution in an order that attempts to maximize some objective function. Hemmati et al. [6] and Henard et al. [10] investigated ways to select an affordable subset with maximum fault detection rate by maximizing diversity among configurations using the

dissimilarity measure. The results obtained in those papers suggested that two dissimilar configurations have a higher fault detection rate than similar ones since the former ones are more likely to cover more components than the latter. Hemmati et al. [6] proposed similarity-based techniques to reduce the cost of model-based test case selection. Hemmati et al [6] also stated that the similarity functions divided by two, which are set-based and sequence-based. Henard et al. [10] sample and prioritize products at the same time by employing a search-based approach to generate products based on similarity among them. Al-Hajjaji et al. [7] propose a similarity-based product prioritization for SPL testing. The prioritization selects the next configuration to be tested based on the similarities between itself and previous tested products. If one variant has been tested, the following configuration is selected, such that it has the minimum similarity with all previous tested configurations. Fang et al. [9] introduced several similarity-based test case prioritization techniques based on the edit distances of ordered sequences. Their work show an increase toward the fault detection rate and effectiveness in detecting faults in loops.

This paper focus toward the fault detection rate as the objective function of the prioritization technique. To evaluate how quick faults are detected during testing, the Average Percentage of Faults Detected (APFD) metric [11], [13], [14] are used. The APFD metric measures the weighted average of the percentage of faults detected during the execution of the test suite. A similar objective is pursued by Hemmati et al. [6], Al-Hajjaji et al. [7] and Fang et al. [9]. Zhang et al. [15] used the total and additional prioritization strategies to prioritize based on the total numbers of elements covered per test, and the numbers of additional which is the not-yet-covered elements covered per test to increase the rate of fault detection. As for Sanchez et al. [8] work, they present an approach that can combine combinatorial testing and different prioritization criteria to detect faults faster.

III. Similarity Distance

Similarity distance is a real-valued function that quantifies the similarity between two objects. In testing, a similarity distance is used for comparing similarity between two configurations [8]. The purpose of similarity function is to maximize the diversity of selected configurations. The diversity of configurations is computed by a certain dissimilarity measure between each pair of configurations. Consequently, this will increase the chance of detecting faults as early as possible if the diversity of the configurations is maximized [9]. In this paper, four type of similarity distances are used.

A. Hamming Distance

Generally, Hamming Distance is used to measure the two-binary string. It used to denote the difference between them. For this paper, we used the definition of Hamming Distance by Al-Hajjaji et al. [7]. They define the distance between the two configurations as below:

$$d(ci, cj, F) = 1 - \frac{|ci \cap cj| + |(F/\ ci) \cap (F/\ cj)|}{|F|} \quad (1)$$

Above function is define as ci and cj are the two given configurations that relative to the set of features F. The values of distance between configurations are between the number 0 and 1. The closer the value to 0, the more similar the two configurations. If the value is equal to 1, it indicates that the configurations are completely different from each other.

B. Jaccard Distance

The others name of Jaccard Distance is Jaccard Index and known as Jaccard similarity coefficient. In statistic, it is use to compare the similarity and diversity of sample sets. In this paper, we used the Jaccard distance that is defined by Henard et al. [10]. They define the *d* as a distance measure between two configurations, which are *ci* and *cj*, to evaluate the degree of similarity. The definition is given by:

$$d(ci, cj) = 1 - \frac{ci \cap cj}{ci \cup cj} \quad (2)$$

The resulting distance varies between 0 and 1. More particularly, a distance which equal to 1 indicates that the two considered configurations are completely different. Meanwhile, a distance which equal to 0 denotes that the two configurations are same. It attempts to find similar members from both chosen configurations, and divided with the total members that are not similar between them.

C. Counting Function

The Counting function is used to compare two sets of transitions. It is the simplest way of comparing two sets that have reused. Hemmati et al. [6], define the counting function as *Cnt(ci, cj)* is the number of same members in *ci* and *cj*, divided by the average members in *ci* and *cj*.

$$d(ci, cj) = 1 - \frac{ci \cap cj}{((ci + cj) \div 2)} \quad (3)$$

The *ci* and *cj* are respectively refer to the configurations. The values of distance between configurations are between the number 0 and 1. The closer the value to 0, the more similar the two configurations. If the value is equal to 1, it indicates that the configurations are completely different from each other.

D. Sorensen Dice

The Sørensen-Dice index is a simple way to calculate a measure of the similarity of two strings. The values produced are bounded between 0 and 1. The algorithm works by comparing the number of identical character pairs between the two strings. It is beneficial for ecological community data where justification for its use is primarily empirical rather than theoretical. The Sorensen Dice is defined as below:

$$d(ci, cj) = 1 - \frac{2|ci \cap cj|}{|ci| + |cj|} \quad (4)$$

The ci and cj are referring to the configuration. It attempts to find the same members between the configurations, and divide it by the total members that exist between both chosen configurations.

IV. Prioritization Techniques

Prioritization technique arrange the configurations for

testing depending on the specified objectives or goals. In this paper, the goal is on rate of fault detection. Configuration's rate of fault detection is a measure of how quick a configuration detects fault during testing process [11]. This goal aims to achieve a sequence of configurations to be run in a way that faults are detected as soon as possible. In this section, we consider five prioritization techniques, which all of them had been used by previous researchers that related to similarity-based prioritization.

A. All-yes-config Strategy

This strategy is common in the Linux community to test the configuration with the maximum number of selected features [12]. The general idea for this strategy is to select the configuration that has the maximum number of selected features to be tested first. If more than one configuration has the same maximum number of selected features, we take the first one that we found. The rationale of selecting the configuration with the maximum number of selected features as the first to test is to be assumed to cover most faults, which may exist in an individual feature [7]. The selected configuration is added to a list of prioritized configurations, and removed from a list of remaining configurations.

The next step to select the second configuration that will be added to prioritized list is the configuration with the maximum distance to the first configuration that has been selected. In case of two or more configurations with the same distance value, the first configuration that get this value of distance is selected.

There are two configurations now on the prioritized list. The next step is to arrange the remaining configurations. In this step, the distance for each configuration in the list of remaining configurations respect to all in the list of prioritized configurations are considered. The minimum distance between the configurations are considered and placed for a comparison to search for the maximum distance from these distances. Thus, the configuration that has the maximum distance is selected into prioritized list. The same process is continued until all configurations are ordered. All these steps in ordering the configurations are from the work of Al-Hajjaji et al. [7].

B. Local Maximum Distance Prioritization

This prioritization technique is used by Henard et al. [10] as its ability to cover t-sets. The similarity distances are used for prioritizing the configurations. This approach iterates over the initial unordered list of configurations, looking for the two configurations that share the maximum distance. These two configurations are added to the prioritized list and removed from the unordered list of configurations. In case of two or more configurations with the same distance value, the first configuration that get this value of distance is selected. This process is repeated until all the configurations from unordered list are added to prioritized list.

C. Global Maximum Distance Prioritization

This prioritization technique also included within the Henard et al. [10] work. Generally, this approach selects at each step the configuration which is the most distant to all the configurations already selected during the previous steps. First step is to select two configurations inside the unordered list that sharing the highest distance. These two configurations are the first added to the prioritized list.

The next step is to sum the individual distances from the unordered list, with the other configurations inside the prioritized list. Thus, its giving a value for the set. Then the maximum value is obtained by comparing these set values. The configuration that has the maximum value is added to the prioritized list. This process is repeated until the unordered list is empty.

D. Farthest-first Ordered Sequence (FOS)

This prioritization technique is used by Fang et al. [9], which is based on ordered sequences of program entities. It is also used similarity-based as the distances between configurations. This technique using the minimum strategy. First, choose configurations that have a greatest code coverage between them, which means the highest distance from others configurations. Add them to the prioritized list.

To choose the next configurations to be added, the total distances between each configuration inside unordered list with each configuration inside prioritized list need to be calculated. The new values of set are obtained. Within the values of the set, choose the configuration that inherit the minimum value as the next configurations that need to be added inside prioritized list. In case of two or more configurations with the same distance value, the first configuration that get this value of distance is selected. Repeat the step until all the configurations are ordered.

E. Greed-aided-clustering Ordered Sequence (GOS)

For GOS process, configurations in each cluster are prioritized by the additional greedy algorithm and then the configurations are selected from each cluster according to the ordering. This technique is implemented with similarity based by the work of Fang et al. [9]. To find the first configuration, obtain a pair of configurations which have minimum distance from the others. Then, add the configurations into the prioritized list. To choose the next configurations to be added, additional greedy algorithm is used. The first configuration that has maximum value will be added to the prioritized list. Repeat until all the configurations are ordered.

V. Experiments And Results

Our implementation is about the similarity-based prioritization. Our aim is to detect more faults as soon as possible for the product lines under test. In our evaluation, we focus on the following research questions.

RQ1: Which string distance shows better result in rate of early fault detection?

RQ2: Does different prioritization techniques affect the rate of early fault detection?

We begin by describing our experimental settings and then we explain the experimental results.

A. Experimental settings

In SPL, to generate a set of configurations, a feature model is needed. We used the feature model and generated configurations from *MobilePhone* product line which is created by Al-Hajjaji et al. [7].

The feature models usually represented graphically by feature diagrams [16]. Figure 1 shows an example of feature diagrams of a product line *MobilePhone*. Feature diagrams are used to restrict the variability of a product line as not all combinations of features are valid. A valid combination is

called as configuration [7].

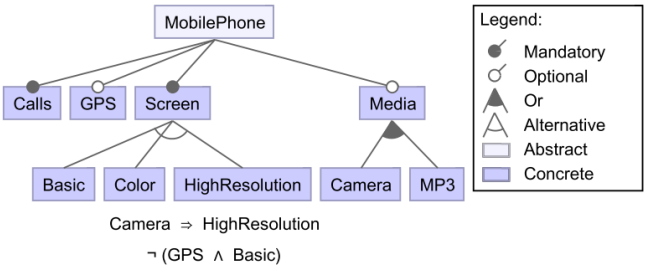

Figure 1: Feature diagram of *MobilePhone* [7]

Table 1 exemplarily lists nine configurations that are created from feature model MobilePhone using pairwise sampling with ICPL [17]. Sampling algorithm typically outputs an ordered list of configurations.

Table 1
Configurations of *MobilePhone* product line [7]

ID	Configurations
C1	{Calls, Screen, Color}
C2	{Calls, GPS, Screen, HighResolution, Media, MP3}
C3	{Calls, Screen, HighResolution, Media, Camera}
C4	{Calls, Screen, Basic}
C5	{Calls, Screen, HighResolution, Media, Camera, MP3}
C6	{Calls, GPS, Screen, Color, Media, MP3}
C7	{Calls, GPS, Screen, HighResolution, Media, Camera}
C8	{Calls, Screen, Basic, Media, MP3}
C9	{Calls, GPS, Screen, HighResolution,}

To measure the effectiveness of our research, we evaluated the ability of the string distances and prioritization techniques to detect faults in the SPL under test. For this purpose, some generated faults are needed. Thus, we used the faults that already generated by Al-Hajjaji et al. [7].

Table 2 shows the distribution of six faults that had been used by Al-Hajjaji et al. [7]. Lastly, to evaluate how quick faults are detected during testing we used the APFD metric. The APFD metric measures the weighted average of the percentage of faults detected during the execution of the test suite. APFD illustrate as the *T* as the test suite which contain a numbers of *n* configurations, and let *F* a set of *m* faults revealed by *T*. Let *TFi* be the position of the first test case in ordering *T'* of *T* which reveals the fault *i*. The equation of APFD is given below:

$$APFD = 1 - \frac{TF1 + TF2 + ... + TFn}{n \times m} + \frac{1}{2n} \qquad (5)$$

APFD value ranges from 0 to 1. A prioritized test suite with higher APFD value has faster fault detection rates than those with lower APFD values.

Table 2
Fault Metric [7]

Configuration	F1	F2	Fault F3	F4	F5	F6
C1		X				X
C2		X	X			
C3				X	X	X
C4	X	X	X			X
C5	X			X		X
C6					X	
C7			X			
C8		X				X
C9						

B. Experiment 1. Implement the similarity distances

This experiment is conducted to apply the similarity distances in Section III for the use of the next experiment. The experimental setup and the results are reported.

1) Experimental Setup

In this experiment, we need to calculate the similarity distances for each of the configuration. Table 1 plays a crucial part to obtain the distances. First, we build a table that contain the *IDs* of configuration. We build four tables due to each distance has different similarity metric.

2) Experimental Results

Table 3 until Table 6 shows the different result for each of similarity distance that we have calculated.

Table 3
Hamming Distance

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	0	0.556	0.444	0.222	0.556	0.334	0.556	0.444	0.334
C2	0.556	0	0.334	0.556	0.222	0.222	0.222	0.334	0.222
C3	0.444	0.334	0	0.444	0.111	0.556	0.111	0.444	0.334
C4	0.222	0.556	0.444	0	0.556	0.556	0.556	0.222	0.337
C5	0.556	0.222	0.111	0.556	0	0.444	0.222	0.334	0.444
C6	0.334	0.222	0.556	0.556	0.444	0	0.444	0.334	0.444
C7	0.556	0.222	0.111	0.556	0.222	0.444	0	0.556	0.222
C8	0.444	0.334	0.444	0.222	0.334	0.334	0.556	0	0.556
C9	0.334	0.222	0.334	0.337	0.444	0.444	0.222	0.556	0

Table 4 until 6 show the calculated distances among each of the configuration. The distances are important due to these values will be used to determine the order of the configuration during prioritization process.

Table 4
Jaccard Distance

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	0	0.714	0.667	0.5	0.714	0.5	0.714	0.667	0.6
C2	0.714	0	0.429	0.714	0.286	0.286	0.286	0.429	0.333
C3	0.667	0.429	0	0.667	0.167	0.625	0.167	0.571	0.5
C4	0.5	0.714	0.667	0	0.714	0.714	0.714	0.4	0.6
C5	0.714	0.286	0.167	0.714	0	0.5	0.286	0.429	0.571
C6	0.5	0.286	0.625	0.714	0.5	0	0.5	0.429	0.571
C7	0.714	0.286	0.167	0.714	0.286	0.5	0	0.571	0.333
C8	0.667	0.429	0.571	0.4	0.429	0.429	0.571	0	0.714
C9	0.6	0.333	0.5	0.6	0.571	0.571	0.333	0.714	0

Table 5
Counting Function

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	0	0.444	0.5	0.667	0.444	0.667	0.444	0.5	0.571
C2	0.444	0	0.727	0.444	0.833	0.833	0.833	0.727	0.8
C3	0.5	0.727	0	0.5	0.909	0.545	0.909	0.6	0.667
C4	0.667	0.444	0.5	0	0.444	0.444	0.444	0.75	0.571
C5	0.444	0.833	0.909	0.444	0	0.667	0.833	0.727	0.6
C6	0.667	0.833	0.545	0.444	0.667	0	0.667	0.727	0.6
C7	0.444	0.833	0.909	0.444	0.833	0.667	0	0.545	0.8
C8	0.5	0.727	0.6	0.75	0.727	0.727	0.545	0	0.444
C9	0.571	0.8	0.667	0.571	0.6	0.6	0.8	0.444	0

Table 6
Sorensen Dice

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	0	0.556	0.5	0.333	0.556	0.333	0.556	0.5	0.429
C2	0.556	0	0.273	0.556	0.167	0.167	0.167	0.273	0.2
C3	0.5	0.273	0	0.5	0.09	0.455	0.09	0.4	0.333
C4	0.333	0.556	0.5	0	0.556	0.556	0.556	0.25	0.429
C5	0.556	0.167	0.09	0.556	0	0.333	0.167	0.273	0.4
C6	0.333	0.167	0.455	0.556	0.333	0	0.333	0.273	0.4
C7	0.556	0.167	0.09	0.556	0.167	0.333	0	0.455	0.2
C8	0.5	0.273	0.4	0.25	0.273	0.273	0.455	0	0.556
C9	0.429	0.2	0.333	0.429	0.4	0.4	0.2	0.556	0

C. Experiment 2. APFD of Similarity-based Prioritization

To answer RQ1 and RQ2, we check the impact on the rate of early fault detection for each of the similarity result obtained with five of the prioritization techniques that are defined in Section IV. The experimental setup and the results are next reported.

1) Experimental Setup

The experimental procedure is to arrange the configurations according to the prioritization technique. To do that, we need to trace a table of the distances row by row, to find which configuration that will be added to the prioritized list. After prioritized list is completed, with Table 2 as reference, we calculate the APFD. Table 7 shows one of the distance that traced manually by using table.

Table 7
Hamming Distance with GOS

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	0	0.556	0.444	0.222	0.556	0.334	0.556	0.444	0.334
C2	0.556	0	0.334	0.556	0.222	0.222	0.222	0.334	0.222
C3	0.444	0.334	0	0.444	0.111	0.556	0.111	0.444	0.334
C4	0.222	0.556	0.444	0	0.556	0.556	0.556	0.222	0.337
C5	0.556	0.222	0.111	0.556	0	0.444	0.222	0.334	0.444
C6	0.334	0.222	0.556	0.556	0.444	0	0.444	0.334	0.444
C7	0.556	0.222	0.111	0.556	0.222	0.444	0	0.556	0.222
C8	0.444	0.334	0.444	0.222	0.334	0.334	0.556	0	0.556
C9	0.334	0.222	0.334	0.337	0.444	0.444	0.222	0.556	0

Table 7 illustrates the process of GOS technique toward the result from Hamming distance. By referring the GOS algorithm, the first configuration that need to be put into prioritized list *P*, is the one that inherit minimum value. Thus, *C4* will be add first because it has smallest value among the other rows. Next configuration will be the *C1*, because the first minimum distance added to the *P* is from the distance between *C4* and *C1*. Now, two configurations that exist in prioritized list are *P= {C4, C1}*.

According to GOS algorithm, the next configuration that will be chosen is the configuration with the maximum value. There are three configurations that have maximum value. In case we have two or more configurations with the same distance value, we select the first configuration that gets this value of distance. Hence, the *C2* (yellow color) is added first as the third configuration inside *P*, followed by *C5* and *C7*. Now, the configurations that remain in a set *C* are *C= {C3, C6, C8, C9}*. Repeat the process until the *C* is empty. Thus, the new order that need to be tested is *P= {C4, C1, C2, C5, C7, C3, C8, C6, C9}*.

The last step is to calculate the APFD for the new order of configurations. Table 8 is created based on the fault metric in Table 2.

Table 8
New order of fault matrix

Configuration	Fault					
	F1	F2	F3	F4	F5	F6
C4	X	X	X			X
C1		X				
C2		X	X			
C5	X			X		X
C7			X			
C3				X	X	X
C8		X				X
C6					X	
C9						

Table 8 contains new faults positions after we prioritized the Hamming distance result by using GOS algorithm. To calculate the APFD, this table is required. The equation of the APFD already given above in Section A in Experimental Setting. The calculation for APFD shown below:

$$APFD = 1 - \frac{1+1+1+4+6+1}{9 \times 6} + \frac{1}{2 \times 9} = 0.796 \quad (6)$$

The TF1 is equal to 1 because the first fault that we found from the first column of table is at the first row of the table. TF is the position of the fault that first to emerge. Thus, it is 1 because the first fault that we encounter first is located at the first row. Next, we look at the second column, which is F2. At which row that the first fault, emerge. Again, the first fault we encounter is at the first row. It goes the same way as for F4 and F6. For the F4 column, the TF4 is equal to 4 because the first fault that can be found is at the row four. Same concept also with the F5.

2) Experiment Results

The result will be four new sets of configurations that had been reorder by each of the prioritization technique used. Table 9 shows the APFD result for each similarity distance with five prioritization techniques. Further explanation related to this result will be discuss in the discussion section below.

Table 9
APFD Result

Similarity Distance	Prioritization Technique				
	All-yes-config	Local Max.	Global Max.	FOS	GOS
Hamming	0.759	0.759	0.778	0.611	0.796
Jaccard	0.759	0.759	0.759	0.741	0.796
Counting Function	0.759	0.759	0.685	0.63	0.796
Sorensen-Dice	0.759	0.759	0.685	0.63	0.796

D. Discussion

In this section, we discuss about our obtained results. Our first results are in Table 3 until Table 6. As we can see, these distances between configurations are needed first. Prioritization cannot be done without them. The value for each pair of configuration is in between 0 and 1. The value indicates the similarity between them. The value that near to 1 indicates that the more dissimilar the configurations are, and vice versa. From the APFD result in Table 9, it shows that the usage of similarity distance can affect the fault detection rate. Jaccard distance shows the highest reading among the others similarity distances. APFD value ranges from 0 to 1. Sanchez et al. [8] state that prioritized test suite

with higher APFD value has faster fault detection rates than those with lower APFD values. Thus, Jaccard distance shows promising result in detecting fault faster, compared to others. Hemmati et al. [6] work mentioned that Jaccard distance is more practical due to its easier to use because it does not require any parameter settings. The work also mentioned that the Jaccard distance has low variation, low cost, and high effectiveness. However, the Jaccard distance does not shine most when ongoing the FOS prioritization technique.

As for the five prioritization techniques used, the GOS prioritization technique shows the highest APFD value. The value shows the same, which is 0.796. According to Fang et al. [9], GOS algorithm is one of the group that use minimum distance. The results from Jiang et al. [18] indicate that the group using minimum strategy has the highest rate of fault detection.

VI. CONCLUSION

Product line testing consumes a lot of time. Every testers expectation is to increase probability of detecting faults as soon as possible for the product line under test. Therefore, several approaches have been proposed to prioritize products to ensure the earlier products have a higher probability to contain faults. With similarity-based prioritization, the products prioritized based on similarity of their features. We evaluate similarity-based prioritization by using four different similarity metrics, with all five different prioritization techniques. We evaluated all of them in term of rate of early fault detection. The results show that the difference between the effectiveness within the similarity metrics and the prioritization techniques. The results showed that the Jaccard distance has better rate of fault detection among the three others similarity distance. For the prioritization technique, GOS algorithm appear the best.

As future work, we plan to work with GOS algorithm to improve the fault detection rate. We plan to enhance the previous algorithm used to achieve a better APFD results than the results shown on this paper.

ACKNOWLEDGMENT

Warm thanks for all the anonymous readers and reviewers who read and review this paper. Comments and suggestions are most welcome for improving the future works of this paper. This research work relatively supported by Dr. Mohd Adham bin Isa research's grant by Ministry of Education Malaysia under the Fundamental Research Grant Scheme (FRGS) with vot. R. J130000.7828.4F836.

REFERENCES

[1] P. Clements, and L. Northrop, *Software product lines: Practices and Patterns.* USA: Addison-Wesley, 2002, pp. 5-6.

[2] D. M. Weiss, "The Product Line Hall of Fame," in *Proceedings of the International Software Product Line Conference (SPLC)*, San Francisco, 2008, pp. 395-395.

[3] G. Perrouin, S. Oster, S. Sen, J. Klein, B. Budry, and Y. le Traon, "Pair-wise testing for software product lines: comparison of two approaches," *Software Quality Journal*, vol. 20, no. 3-4, pp. 605-643, Sep. 2012.

[4] C. Catal and D. Mishra, "Test case prioritization: a systematic mapping study," *Software Quality Journal*, vol. 21, no. 3, pp. 445-478, Sep. 2013.

[5] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test case prioritization: An empirical study," in *Proceedings IEEE International Conference on Software Maintenance*, Oxford, 1999, pp. 179-188.

[6] H. Hemmati, and L. Briand, "An industrial investigation of similarity measures for model-based test case selection," in *Symposium IEEE 21st International on Software Reliability Engineering (ISSRE)*, San Jose, 2010, pp. 141-150.

[7] M. Al-Hajjaji, T. Thüm, M. Lochau, J. Meinicke, and G. Saake, "Effective product-line testing using similarity-based product prioritization," *Software & Systems Modeling*, vol. 16, pp. 1-23, Dec. 2016.

[8] A. B. Sánchez, S. Segura, and A. Ruiz-Cortés, "A comparison of test case prioritization criteria for software product lines," in *Conference IEEE Seventh International Conference on Software Testing, Verification and Validation (ICST)*, Cleveland, 2014, pp. 41-50.

[9] C. Fang, Z. Chen, K. Wu, and Z. Zhao, "Similarity-based test case prioritization using ordered sequences of program entities," *Software Quality Journal*, vol. 22, no. 2, pp. 335-361, Jun 2014.

[10] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. Le Traon, "Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines," *IEEE Transactions on Software Engineering*, vol. 40, no. 7, pp. 650-670, Jul 2014.

[11] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE transactions on software engineering*, vol. 28, no. 2, pp. 159-182, Feb 2002.

[12] C. Dietrich, R. Tartler, W. Schröder-Preikshat, and D. Lohmann, "Understanding linux feature distribution," in *Proceedings of the 2012 workshop on Modularity in Systems Software*, Potsdam, 2012, pp. 15-20.

[13] X. Qu, M. B. Cohen, and K. M. Woolf, "Combinatorial interaction regression testing: A study of test case generation and prioritization," in *Conference IEEE International Conference on Software Maintenance (ICSM)*, Maison Internationale, 2007, pp. 255-264.

[14] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on software engineering*, vol. 27, no. 10, pp. 929-948, Oct 2001.

[15] L. Zhang, D. Hao, L. Zhang, G. Rothermel, and H. Mei, "Bridging the gap between the total and additional test-case prioritization strategies," in *Proceedings of the 2013 International Conference on Software Engineering*, San Francisco, 2013, pp. 192-201.

[16] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, *Feature-oriented domain analysis (FODA) feasibility study*. Pittsburgh, PA: Carnegie-Mellon University, 1990.

[17] M. F. Johansen, Ø. Haugen, and F. Fleurey, "An algorithm for generating t-wise covering arrays from large feature models," in *Proceedings of the 16th International Software Product Line Conference*, Salvador, 2012, pp. 46-55.

[18] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse, "Adaptive random test case prioritization," in *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, Auckland, 2009, pp. 233-244.