Enhancing Similarity Distances Using Mandatory and Optional forEarly Fault Detection

Safwan Abd Razak, Mohd Adham Isa, Dayang N.A Jawawi
Department of Software Engineering, Faculty of Computing, Universiti Teknologi Malaysia, Malaysia

Article Info

Article history:

Received May 3, 2018
Revised Jun 4, 2018
Accepted Jun 18, 2018

Keywords:

Average percentage of faults detected (APFD)
Prioritization
Similarity distances algorithms
Software product lines

ABSTRACT

Software Product Line (SPL) describes procedures, techniques, and tools in software engineering by using a common method of production for producing a group of software systems that identical from a shared set of software assets. In SPL, the similarity-based prioritization can resemble combinatorial interaction testing in scalable and efficient way by choosing and prioritize configurations that most dissimilar. However, the similarity distances in SPL still not so much cover the basic detail of feature models which are the notations. Plus, the configurations always have been prioritized based on domain knowledge but not much attention has been paid to feature model notations. In this paper, we proposed the usage of mandatory and optional notations for similarity distances. The objective is to improve the average percentage of faults detected (APFD). We investigate four different distances and make modifications on the distances to increase APFD value. These modifications are the inclusion of mandatory and optional notations with the similarity distances. The results are the APFD values for all the similarity distances including the original and modified similarity distances. Overall, the results shown that by subtracting the optional notation value can increase the APFD by 3.71% from the original similarity distance.

Corresponding Author:

Safwan Abd Razak,
Department of Software Engineering, Faculty of Computing,
Universiti Teknologi Malaysia, Malaysia.
Email: safwan7@live.utm.my

1. 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. Testing a product line is referring to extraction from a set of products and test every single 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.

Prioritization techniques arrange test cases for implementation in a position that attains to improve their effectiveness in achieving certain performance goals [4-5]. Various goals can be specified. For examples, the software testers may want to arrange the test cases in an position that can attain full code coverage as soon as possible or in an order that can improve the rate of fault detection. State a goal first, then several ordering criteria can be considered. For an example, set the improvement rate of fault detection as a goal. Software testers could arrange the test cases by the presumed dispose error of the component under test or they also could position the test cases depending on the total of faults identified by the previous executed test cases.

There are many types of the prioritization techniques such as string-based, requirement-based, fault-based, coverage-based, and history-based. Each type has different strategies in prioritizing the test cases. This paper focus on the similarity distances algorithms which are used within string-based prioritization. We explore the applicability of the similarity distance with the prioritization technique in improving early fault detection rate. Four type of similarity distances are used. Those are Hamming distance, Jaccard distance, Counting function, and Sorensen-Dice. The reason we used prioritization based on similarity distances is that it has higher fault detection rate and higher feature coverage [6]. Our porposed work are the enhancement of these four similarity distances algorithms.

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. Each of these similarity distances then are prioritized with five different prioritization techniques. The similarity distances between the configurations are calculated. The distances obtained are used to prioritize the configurations. Finally, we calculate the average percentage of faults detected.

2. SIMILARITY DISTANCES

Similarity function is introduced to maximize the diversity of configurations. On the other hand, prioritization technique organizes the configurations for implementation in a position that strives to maximize some objective function. Hemmati *et al.* [6] and Henard *et al.* [8] used dissimilarity measure to maximize diversity among configurations. They explored methods to determine a subset that affordable, that possess maximum rate of fault detection. Results from those papers advocated that high fault detection rate can be achieved from two dissimilar configurations rather than similar ones. This due to the earlier ones are more likely to cover more components than the latest. In this section, we describe the four similarity distances that we used.

2.1. Jaccard Distance

The Jaccard Distance is also known as Jaccard similarity coefficient. In statistic, it is used in comparison of sample sets that involve diversity and similarity. The Jaccard model is a similar measure based on common words [9]. In this paper, we used the Jaccard distance that is defined by Henard et al. [8]. 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:

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

The distance is between 0 and 1. Specifically, the configurations are totally different from one another if the value is equal to 1. Meanwhile, a distance which the value is 0 specifies both 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.

2.2. 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:

$$d(ci,cj,F) = 1 - \frac{|ci \cap cj| + |(F/\ ci) \cap (F/\ cj)|}{|F|} \tag{2}$$

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. The configurations are totally different from one another if the value is equal to 1.

2.3. Counting Function

The Counting function is used to compare two sets of transitions. It is the straightforward method to compare two reused sets. 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*.

$$d(ci,cj)=1-\frac{ci\bigcap cj}{\left(\left(ci+cj\right)\div 2\right)} \tag{3}$$

The ci and cj are respectively refer to the configurations. The distance values among configurations are bounded by number 0 and 1. The closer the value to 0, the more similar the two configurations. The configurations are totally different from one another if the value is equal to 1.

2.4. Sorensen Dice

The Sørensen-Dice index is a simple way to calculate a measure of the similarity of two strings. The values produced also are bounded between 0 and 1. The algorithm works by comparing between two strings the total of same character pairs. 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:

$$d(ci,cj)=1-\frac{2\left|ci\bigcap cj\right|}{\left|ci\right|+\left|cj\right|} \tag{4}$$

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.

3. SIMILARITY DISTANCES ENHANCEMENT

In this section, we present our proposed work which are the enhancement of similarity distances algorithms with the addition of the feature model (FM) constraints in SPL.

3.1. Feature Model Notations

In software development, a feature model is a solid potrayal of entire products from the SPL in term of features. During product line development process, feature models are widely used as input to produce other assets. These assets are the description of architecture, documents, or parts of code. The graphical representation of a feature model is called a feature diagram [10].

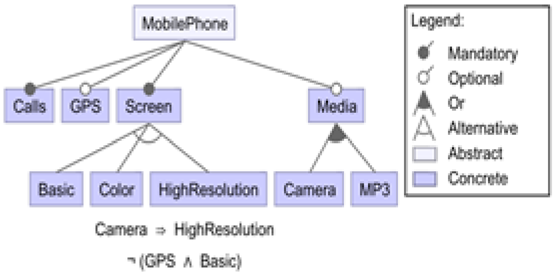

Figure 1. Feature diagram of product line MobilePhone [7]

Feature diagram can be described as a tree where others than root feature, each feature has a parent feature. One or more features can be decomposed from each feature, except for terminal features. Feature diagram's notation is the rules when selecting features to derive a product.

As shown in Figure 1, there are four types of notations. Those are connections between child features or sub-features with their parent feature. Those groups are:

1. Mandatory: Child feature is required
2. Optional: Child feature is optional
3. Or: At least one of the sub-features must be selected
4. Alternative: One of the sub-features must be selected

The features are either abstract or concrete. The feature is concrete if implementation artifacts are mapped to a feature, such as the *Calls* feature. Apart from that, the feature is abstract, such as the *MobilePhone* feature. On top of that, among the features there are additional dependencies which cannot be explained with a hierarchical structure. Cross-tree constraints for example. Most common used of cross-tree constraints are:

1. *A* requires *B*: In a product, selection of *A* suggests the selection of *B*. Example, the *Camera* feature selection in a mobile phone suggests the *HighResolution* feature selection.
2. *A* excludes *B*: In a product, *A* and *B* must not be in the same part. For instance, same mobile phone cannot support both *GPS* and *Basic* features.
3. By using logical operators ↔, ∧, ¬, →, and ∨, additional constraints can be defined as propositional formulas.

Thus, the input of the similarity-based prioritization will be the selection of features from a feature model, which are called as configurations.

3.2. Enhanced Similarity Distances

We consider the feature model notations in our work to improve the existing similarity distance algorithm. For our research, we only selected two feature model notations which are mandatory and optional. This is because in feature model, mandatory and optional are the crucial notations on every feature model. It is compulsory for the feature models to have both notations. Without them, the Or and Alternative notations cannot be used.

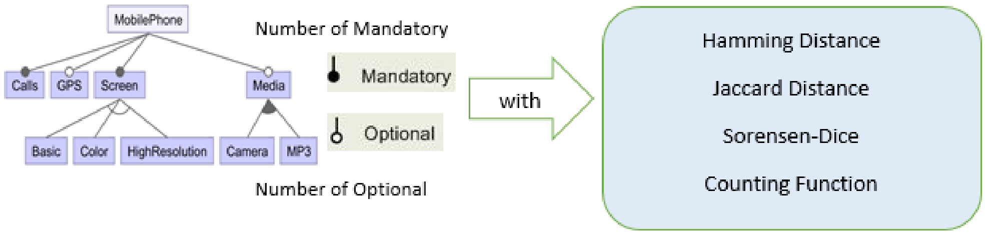

Figure 2. Overview of proposed work

As shown in Figure 2, we want to consider the Mandatory and Optional notation from feature model inside similarity distances. In this section, we only highlighted Jaccard distance. There are four modifications that we have tried with the similarity distances listed in Section 2. The modificatios done are given as follows.

Modification 1: Addition of Mandatory

We modify the similarity distance by adding one variable that represent Mandatory notation. The Mandatory notation is the main notation for all feature models. The reason is that it represents the mandatory feature(s) of the product. Even Optional notation cannot surpass the importance of Mandatory since without the Mandatory, there will be no product exist. The reason we consider adding the Mandatory notation into the algorithm is that we want to increase the chances of configurations that embed these mandatory features to be selected first. If there any fault in it, the tester can detect much faster. Moreover, it will be a threat toward the product if the tester missed some faults that involved with mandatory feature. We define Jaccard distance with the addition of Mandatory variable as:

$$d(c_i, c_j) = 1 - \frac{(c_i \cap c_j) + m}{(c_i \cup c_{j)} + m}$$

Where *m* is the number of Mandatory notations from the feature model used. If there are two Mandatory notations inside the feature model, the value of *m* is 2. These distances are named as Addition Hamming Mandatory (AHM), Addition Jaccard Mandatory (AJM), Addition Counting Function Mandatory (ACFM), and Addition Sorensen-DIce Mandatory (ASDM).

Modification 2: Addition of Optional

We modify the similarity distance by adding one variable that represent Optional notation. Optional notation is one of the notation that represent variable features. The variable features used to express variability. Inherently, reusable software contains more variability [11]. Thus, it is important to focus on only optional feature. This due to this feature will be reusable later for the new product. There are chances also that this optional feature will become mandatory in the future. Therefore, it is wise for the tester to solve the fault earlier before it spread out when releasing new products. We define Jaccard distance with the addition of Optional variable as

$$d(c_i, c_j) = 1 - \frac{(c_i \cap c_j) + o}{(c_i \cup c_{j)} + o}$$

Where o is the number of Optional notations from the feature model used. If there are two Optional notations inside the feature model, the value of o is 2. These distances are named as Addition Hamming Optional (AHO), Addition Jaccard Optional (AJO), Addition Counting Function Optional (ACFO), and Addition Sorensen-DIce Optional (ASDO).

Modification 3: Addition of Mandatory and Optional

We modify the similarity distance by adding two variable which represent Mandatory and Optional notations. The product line is about commonality and variability. Features that describe only one of them cannot be useful because the individual instances of valid configurations probably do not describe the system in enough detail [12]. Thus, we consider adding both notations inside the algorithm. We define Jaccard distance with the addition of Mandatory and Optional variables as

$$d(c_i, c_j) = 1 - \frac{(c_i \cap c_j) + m + o}{(c_i \cup c_{j)} + m + o}$$

Where o is the number of Optional notations and m is the number of Mandatory notation from the feature model used. If there are two Mandatory notations inside the feature model, the value of m is 2. Same concept used for Optional notations. These distances are named as Addition Hamming Mandatory Optional (AHMO), Addition Jaccard Mandatory Optional (AJMO), Addition Counting Function Mandatory Optional (ACFMO), and Addition Sorensen-DIce Mandatory Optional (ASDMO).

Modification 4: Subtraction of Optional

We modify the similarity distance by subtracting one variable that represent Optional notation. We define Jaccard distance with the subtraction of Optional variable as

$$d(c_i, c_j) = 1 - \frac{(c_i \cap c_j) - o}{(c_i \cup c_{j)} - o}$$

Where o is the number of Optional notations from the feature model used. If there are two Optional notations inside the feature model, the value of o is 2. These distances are named as Subtract Hamming Optional (SHO), Subtract Jaccard Optional (SJO), Subtract Counting Function Optional (SCFO), and Subtract Sorensen-DIce Optional (SSDO).

4. EXPERIMENTAL SETUP

Our implementation is about the similarity-based prioritization. Our aim for the product lines under test is to detect more faults within a short time.

4.1. Generate Configurations

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]. Feature diagrams represent the feature models graphically. Example of feature diagram can be seen in Figure 1 which is for *MobilePhone*. Feature diagrams often used to limit the product line variability. This due to not all combinations of features are valid. Combination that valid is called as configuration [7].

By using pairwise sampling with ICPL [13], nine configurations inside Table 1 are established from *MobilePhone* feature model. The ordered list of configurations is generated by using sampling algorithm.

Table 1. *MobilePhone* Configurations [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

4.2. Implement Similarity Distance

Next step is to apply the similarity distance algorithm. Table 1 plays a crucial part to obtain the distances. All configurations inside Table 1 are used to calculate the distances between the configurations. Table 2 shows one of the generated distances between the configurations.

Table 2 shows the calculated distances among each of the configuration by using the Jaccard distance. The distances are important due to these values will be used to determine the order of the configuration during prioritization process.

Table 2. Generated Distances

	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

4.3. Prioritized Configurations

After the distances are determined, we proceed to arrange the configurations according to the prioritization techniques. 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. We used five prioritization techniques in our work which are All-Yes-Config (AYC), Local Maximum Distance (LMD), Global Maximum Distance (GMD), Farthest-first Ordered Sequences (FOS), and Greed-aided Ordered Sequences (GOS). We used Table 2 as reference to trace the flow of one of the prioritization technique.

Table 3 illustrates the process of GOS technique toward the result from Jaccard 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}.

Table 3. Jaccard Distance with GOS

	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	$\boxed{\mathbf{0.5}}$	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

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. We choose the first configuration in case same distance value is possess by two or more configurations. Thus, the *C2* (bold without square) 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, C9, C6}*.

4.4. Calculate Average Percentage Faults Detected (APFD)

The effectiveness of our research can be measured by evaluation within the ability of the string distances and prioritization techniques in the SPL under test to detect faults. Generated faults are needed for this purpose. Thus, we used the faults that already generated by Al-Hajjaji *et al.* [7].

Table 4 displays the distribution of six faults that had been used by Al-Hajjaji et al. [7]. Lastly, APFD metric used to appraise how quick faults are detected during testing. The APFD metric computes the average weight from percentage of faults detected while executing the test suite. APFD illustrate as the *T* as the test suite which contain a numbers of *n* configurations, and *F* is a set of *m* faults exposed by *T.* Make *TFi* exist as the position of the first test case in *T'* of *T* order which exposes the fault *i.* The equation of APFD is given as:

Table 4. Fault Matrix [7]

Configuration	Faults					
	F1	F2	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						

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

The final step is to calculate the APFD for the new order of configurations. Table 5 is created based on the fault metric in Table 3.

Table 5 contains new faults positions after we prioritized the Jaccard distance result by using GOS algorithm. To calculate the APFD, Table 5 is required. The value is between 0 to 1. High APFD value from prioritized test suite has faster fault detection rates than those with low APFD values. The calculation for APFD shown as:

Table 5. New Configurations Order with Fault Matrix

Configuration	Faults					
	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
C9						
C6					X	

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

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.*

5. RESULTS AND ANALYSIS

In this section, the APFD result for each similarity distance with five prioritization techniques are shown.

Table 6 shown that the similarity SHO presents the highest APFD value which is 0.83333 by using GMD. The distance's values between configurations are from 0 until 1. The closer the value to 0, the more similar the two configurations. If the value is equal to 1, it shows that both configurations are totally different. The second highest also are from the SHO, which is 0.814815 by using LMD and GOS. By using SHO with other prioritization techniques also shows that in overall, the APFD values are still high and consistent compared with Hamming original and others modified Hamming distances.

Table 6. Hamming Distance APFD

	AYC	LMD	GMD	FOS	GOS
H	0.759	0.759	0.778	0.611	0.796
AHM	0.759	0.759	0.778	0.611	0.796
AHO	0.759259	0.703704	0.703704	0.648148	0.777778
AHMO	0.759	0.759	0.704	0.611	0.778
SHO	0.796296	0.814815	0.833333	0.796296	0.814815

Table 7 shown that the SJO presents the highest APFD value. The highest value is shared between GMD and FOS. The second highest value also from the SJO with LMD and GOS. By using SHO with other prioritization techniques also shows that in overall, the APFD values are still high and consistent compared with original Jaccard and others modified Jaccard distances.

Table 7. Jaccard Distance APFD

	AYC	LMD	GMD	FOS	GOS
Jaccard	0.759	0.759	0.759	0.741	0.796
AJM	0.778	0.759	0.722	0.611	0.796
AJO	0.759259	0.703704	0.759259	0.740741	0.666667
AJMO	0.759	0.704	0.63	0.685	0.778
SJO	0.796296	0.814815	0.833333	0.833333	0.814815

Table 8 shown that the similarity SSDO dominates the highest APFD value which is 0.83333 by using FOS. The APFD value shown drastic decreased when using SHO with GMD. Still, by using LMD and GOS, SSDO maintained the second highest APFD value. Overall, the APFD values by using SSDO are still high and consistent compared with original Sorensen-Dice and other modified Sorensen-Dice distances.

Table 8. Sorensen Dice APFD

	AYC	LMD	GMD	FOS	GOS
Sore-Dice	0.759	0.759	0.685	0.63	0.796
ASDM	0.759	0.759	0.685	0.63	0.796
ASDO	0.759259	0.703704	0.648148	0.740741	0.666667
ASDMO	0.759	0.704	0.63	0.741	0.778
SSDO	0.796296	0.814815	0.759259	0.833333	0.814815

Table 9. Cnt. Funtion APFD

	AYC	LMD	GMD	FOS	GOS
Cnt. Func	0.759	0.759	0.685	0.63	0.796
ACFM	0.778	0.759	0.722	0.611	0.796
ACFO	0.759259	0.703704	0.666667	0.740741	0.666667
ACFMO	0.759	0.704	0.63	0.741	0.778
SCFO	0.796296	0.814815	0.759259	0.833333	0.814815

By using SCFO, the highest APFD value is gained by using the FOS. Second highest value by using LMD and FOS. With SCFO, the whole APFD value are still high and consistent compared with original Cnt. Function and other modified Cnt. Function distances.

6. DISCUSSION

As can be seen in Table 6 until Table 9, similarity distances that undergo modifications which involved Mandatory and Optional notations produced different APFD values. Some values are worst that the original and others are slightly better. The worst value is 0.611 and the best value is 0.833333. Obviously, similarity distances that undergo **Modification 4** yield the best APFD values by using GMD and FOS. According to our previous results, we stated that Jaccard distance has better rate of fault detection from the others and GOS technique outwits other techniques [14]. Our current results denote that Jaccard distance still the better one. Results from Table 7 demonstrate that two best values obtained by using GMD and FOS. On the other hand, other similarity distances only produced best value either by using GMD or FOS. Jaccard distances widely used is statistic for measuring sample sets similarity and diversity [15]. One of the reason the Jaccard distance is effective because it extremely sensitive to small samples sizes especially with very small samples or data sets [16]. In our work, we modified the Jaccard distances by removing the Optional features from the feature model, which is the sample. Therefore, by reducing the sample sizes can increased the APFD value. Another reason to use Jaccard is that it has low variation, low cost, and high effectiveness [6]. On the other hand, our current results shown that GOS technique cannot exceed the best value when combined with any **Modification 4** distances. The results indicate that GOS mostly perform well than the other techniques. We can see from higher APFD values that produced by GOS than AYC, LMD, GMD and FOS with **Modification 1** and **Modification 3**. Supposedly GOS can outperforms other techniques used for **Modification 4** distances. This due to GOS is one of the group that use minimum distance strategy and by using minimum strategy should produce high rate of fault detection [17]. Although minimum strategy can increase the APFD, it also can cause the optimization problem [18]. This due to nature of greedy algorithm used in GOS. Both GMD and FOS techniques consider calculating the distances between unorder list with prioritized list while GOS only consider the unorder list. This can make the GOS overlooks the configurations that may contain faults. Thus, right approach need to be designed to solve the problem.

7. CONCLUSION

Product line testing consumes a lot of time. Every testers expectation during testing is to detect faults as soon as possible. Therefore, several approaches related to prioritize products have been proposed to ensure higher probability of faults are detected in the earlier products. One of them is similarity-based prioritization. In this paper, we have proposed enhancement for the similarity distances that have been used in SPL to improve early fault detection rate. We utilize feature model notations (Mandatory and Optional) into similarity distance algorithms that been used in SPL field. This due to enable the configurations that have important features to be tested first for any existing faults. Finding faults early within important features are cost friendly. Our results express improvement in early fault detection. By considering the subtraction of Optional notation into the similarity distances can improve the APFD value.

For future work, we plan to improve GOS so that it can tunes with the enhancement made and outperform the current results. Plus, more feature models need to be used to find that whether our work still effective on various size of product line, from small to large sizes. Furthermore, different feature models simulate different faults. Thus, there exist an uncertainty toward the APFD results and we tend to find about that.

ACKNOWLEDGEMENTS

Warm thanks to 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] Clements P, Northrop L. Software product lines: practice and patterns. Pittsburgh: Addison-Wesley Professional. 2001.

[2] Weiss DM. The Product Line Hall of Fame. Proceedings of International Software Product Line Conference (SPLC). San Francisco. 2009: 395-395.

[3] Perrouin G, Oster S, Sen S, Klein J, Budry B, le Traon Y. Pair- wise testing for software product lines: comparison of two approaches. *Software Quality Journal*. 2012; 20(3-4): 605-643.

[4] Catal C, Mishra D. Test case prioritization: a systematic mapping study. Software Quality Journal. 2013; 21(3): 445-478.

[5] Rothermel G, Untch RH, Chu C, Harrold MJ. *Test case prioritization: An empirical study*. Proceedings of IEEE International Conference on Software Maintenance (ICSM'99). Oxford. 1999: 179-188.

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

[7] Al-Hajjaji M, Thüm T, Lochau M, Meinicke J, Saake G. Effective product-line testing using similarity-based product prioritization. *Software & Systems Modeling*. 2016; 16(22): 1-23.

[8] 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 Transactions on Software Engineering*. 2014: *40*(7): 650-670.

[9] Boubacar A, Niu Z. Valuing Semantic Similarity. *Indonesian Journal of Electrical Engineering and Computer Science*. 2014: 12(8): 6361-6368.

[10] Kang KC, Cohen SG, Hess JA, Novak WE, Peterson AS. Feature-oriented domain analysis (FODA) feasibility study. PHD Thesis. Pittsburgh: Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst:1990.

[11] Estublier J, Vega G. Reuse and variability in large software applications. *ACM SIGSOFT Software Engineering Notes*. 2005: *30*(5): 316-325.

[12] Osis J. Model-driven domain analysis and software development: Architectures and functions: Architectures and functions. Latvia: IGI Global. 2010.

[13] Rothermel G, Untch RH, Chu C, Harrold MJ. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering*, 2001: *27*(10): 929-948.

[14] Razak SA, Isa MA, Jawawi DNA. A Comparison on Similarity Distances and Prioritization Techniques for Early Fault Detection Rate. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*. 2017; *9*(3-3): 89-94.

[15] Jaccard P. Bull Soc Vaud Sci Nat. In: Hanrahan G, Gomez FA. *Editors.* Chemometric methods in capillary electrophoresis. New Jersey: John Wiley & Sons: 2009: 223-270.

[16] Milligan GW, Schilling DA. Asymptotic and finite sample characteristics of four external criterion measures. *Multivariate Behavioral Research*. 1985: 20(1): 97-109.

[17] Jiang B, Zhang Z, Chan WK, Tse TH. (2009, November). *Adaptive random test case prioritization*. In Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering. Washington. 2009: 233-244.

[18] Nguyen PH, Wang D, Truong TK. A new hybrid particle swarm optimization and greedy for 0-1 knapsack problem. *Indonesian Journal of Electrical Engineering and Computer Science*. 2016: 1(3): 411-418.