An Experiment of Different Similarity Measures on Test Case Prioritization for Software Product Lines

Muhammad Sahak, Dayang N. A. Jawawi and Shahliza A. Halim
Faculty of Computing, Universiti Teknologi Malaysia, 81310 Johor Bahru, Johor, Malaysia.
muhammadbsahak@gmail.com

***Abstract*—Software product line (SPL) engineering paradigm is commonly used to manage variability and commonalities of business applications to satisfy a specific need or goal of a particular market. However, due to time and space complexity, combinatorial interaction testing (CIT) has been suggested to reduce the size of test suites. Although CIT is known as a promising approach to overcome these problems, there are still issues such as combinatorial explosion of features, which drains budget allocated for testing. Therefore, test case prioritization (TCP) is preferred to gain a better result in terms of producing an efficient detection of faults. Among prioritization techniques used in regression testing is similarity-based test case prioritization. Similarity-based test case prioritization rearranges test cases through calculation of distance between test cases using similarity measures. Result from the use of similarity measures in test case prioritization contributes to a much better testing process. This paper provides a comparison of selected similarity measures to investigate the feasibility and suitability of similarity measures to be used in SPL through experimentation. Jaccard, Hamming, Jaro-Winkler, Cosine similarity, Counting, and Sorensein distances have been chosen as similarity measures in this study. The result showed Jaro-Winkler as the best similarity measure with an 84.96% Average Percentage of Faults Detected (APFD) value across eight feature models. The study offers insights on similarity measures in SPL context. Further, the paper concludes with suggestions on room for improvement, which could be achieved through experimentation and comparison studies.**

***Index Terms*—Similarity-based; Similarity Measure; Software Product Line Testing; Test Case Prioritization.**

I. INTRODUCTION

Software Product Line (SPL) engineering is founded on the concept of reusability of products from the same family, which can be systematically reused either as common assets or only shared by a subset of the family [1]. Many software organizations modify their development process from single system to SPL to take advantage of reduction of time, cost, and effort to market, while significantly increases the quality of the derived products. SPL can be shown in graphical fashion using Feature Model (FM), which describes inter-relationships between features. FM helps in modelling commonalities and variability of all products within a product family for product derivation process, which is known as configuration process. In this process, a selection of desirable features from FM to be developed for final application only allows a valid combination of features to be formed, which is known as configuration [2]. One of common quality assurance measures in SPLs is SPL testing. The difference between testing a single system and SPL is a single system only considers a single product at a time, whereas SPL considers entire SPL products to be tested. Due to this, SPL needs a systematic testing process due to commonalities and variabilities of features. SPL testing process struggles with complexity when the number of configurations (products) increases exponentially as the number of features grows linearly, which is known as combinatorial explosion. Resources allocated for SPL testing might be exhausted before testing completes and the faults might be left undetected. Thus, this prompts the need for a new method to overcome these challenges. A promising method to deal with these challenges comprises regression testing which is capable of reducing the number of test artifacts in single system either through minimization, selection, or prioritization [15]. Test case prioritization (TCP) is chosen to overcome SPL testing challenges as it considers best sequence of test cases to be tested, which may help in reducing the effort of testing.

Moreover, TCP has been adapted in recent SPL works in [1], [2], [3], and [6]. TCP based on similarity measures is no stranger in test case prioritization method as indicated in [3], [6], and [12]. This approach, known as similarity-based prioritization aims to reorder test cases in terms of dissimilarity values for an SPL configuration to achieve a certain criteria. A graphic depiction of a similarity-based prioritization approach is shown in Figure 1. Similarity-based measures work on the assumption that most dissimilar test cases will generate most dissimilar configurations, which could produce more errors compared to similar ones [3]. Similarity values of a configuration are between zero and one, with zero indicating the configuration as similar, whereas one indicating the configuration as completely different, although this varies according to similarity measures' formula. Typically, the input for an SPL similarity-based approach is a set of sampled configurations which is generated either through a domain expert or a sampling algorithm that undergoes combinatorial interaction testing (CIT) such as ICPL, AETG, CASA, and Chavtal, which is also known as test case selection.

The remainder of the paper is organized as follows. Section II summarizes related works on similarity measures in test case prioritization. Section III states the background in SPL testing. Section IV includes experimental process and results based on similarity measures, which are presented and illustrated. Section V includes discussion based on the experiment's results. Section VI deliberates threat to validity. Finally, Section VII draws conclusions and future work.

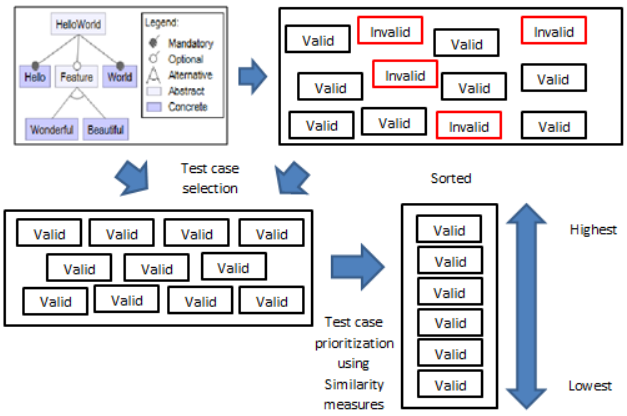

Figure 1: Similarity-based prioritization approach

II. Related Works

In previous studies, various researchers have applied similarity measures in a single system testing process. This includes the work by Ledru et al. [36], which have investigated the use of string distances in test case prioritization and determined the best type of string distances through comparing four classical string distance metrics. The result obtained is a string distance that is feasible to be used. The researchers reported Manhattan distance as the best choice to be used for prioritization purpose. Another work in single system is an empirical study performed on the effects of different similarity measures used for test case prioritization by Wang et al. [37]. The work evaluated the effects of six similarity measures on two similarity-based test case prioritization algorithms. The results obtained by their statistical analysis showed that Euclidian distance is more efficient in finding defects than other similarity measures. Moreover, Cohen et al. [38] carried out a comparison of several string distance metrics in name-matching tasks and concluded that Jaro-Winkler is a fast distance metric in calculation.

Another work by Choi et al. [22] gathered 76 distance measures and binary similarity measures for classification according to hierarchical clustering and performed a study their relationships. Their study provided more insights on various similarity measures that have yet to be used in SPL domain. Furthermore, work by Bilenko and Yurveyvich, [23] discussed similarity functions' importance in learning problem and suggested the creation of a function that can adapt to a particular domain as a suitable area to be researched.

In SPL, there is a rising number of contributions in similarity-based prioritization. Among the work done using similarity-based prioritization technique is by Henard et al. [3]. The work proposed a combination of similarity heuristics and search-based approaches to prioritize test suites. The results indicated that two most dissimilar test cases will generate a higher fault detection rate than similar ones since the former ones are more likely to cover more features than the rest, which leads to more faults detected. Moreover, work by Sanchez et al. [10] conducted a comparison of test case prioritization criteria by dividing the approaches inside TCP to five categories. The categories consist of dissimilarity measures with Jaccard distance selected to be used as dissimilarity prioritization criteria's representative. The experiment investigated whether prioritization criteria presented are effective at improving the rate of early fault detection of SPL test suites, and whether the criteria are able to improve current fault detection rate. The work indicated significant differences in the rate of early fault detection provided by different prioritization criteria.

Another work by Devroey et al. [28] carried out an investigation on dissimilarity-based test generation for SPL behavior model and utilized similarity measures such as Hamming, Jaccard, Dice, Anti-dice, and Levenstein. They concluded that Hamming Distance and Jaccard distance as the most efficient similarity measures. More recently, another work by Al-hajjaji et al. [6] proposed a similarity-based prioritization approach to improve early rate of fault detection and interactive coverage between features. The work utilized Hamming distance as a similarity measure and compared it with a default order of sampling algorithms such as ICPL, CASA, and Chavtal, as well as, random order of test suites. Results obtained through their experimentation showed that Hamming distance improved the default order of configuration and indicated current sampling algorithm—default order is already suitable for testing, while modification indeed had improved the result of experiment.

Moreover, test suites generated by sampling algorithm are commonly ordered by using a similarity-based prioritization algorithm. The algorithm helps to decide test cases' new placements in a prioritized test suite to achieve a desired goal. In single system, various works have been done to improve prioritization algorithms such as [14], [35], and [32]. Fang et al. [35] provided a new technique for test case prioritization through an empirical study based on farthest-first ordered sequence (FOS) algorithm and greed-aided-clustering (GOS) algorithm. Their work concluded that their technique was able to find bugs and increased fault detection rate. Furthermore, nearest neighbor algorithm is regarded as the most suitable algorithm for a large dimensionality of data and efficient in generating a low computational overhead [32]. Another work by Wang et al. [37] investigated the effects of similarity measures on two similarity-based algorithms—ART-based prioritization algorithm (ART) and global similarity-based prioritization algorithm. The study concluded Euclidian distance might be a better choice to be used in test case prioritization. In SPL, various types of prioritization algorithms have been used in SPL including [3], [6], and [10]. Work by Henard et al. [3] incorporated local maximum prioritization and global maximum prioritization algorithms to bypass combinatorial explosion in SPL. Whereas, Sanchez et al. [10] used a local maximum prioritization in their work to investigate the best type of test case prioritization technique. On the other hand, Al-Hajjaji et al. [6] proposed an all-yes-config algorithm, which is typically used in Linux community to select the most number of features in a product as the first one to be tested in a prioritized test suite.

Based on the studies analyzed, to the best of the authors' knowledge, there is no extensive comparison on similarity measures carried out by researchers in the past on test case prioritization for SPL. Thus, the current study is motivated to conduct an empirical study to investigate various types of similarity measures, in the context of white-box testing in test case prioritization.

III. BACKGROUND

A. Feature Model

The usage of Feature Model (FM) is commonly used in SPL as a visual representation to describe features that exist along with their relationship between each other. Feature modelling was first introduced by Kang [29] in Feature-Oriented Domain Analysis (FODA), which has been utilized widely in SPL since it supports SPL's development life cycle. Figure 2 shows an Electronic Shopping FM consisting of eight features with their relationships. The relationships that exist in the FM are as follows:

i. *Mandatory* - Catalogue is required as a child feature of E-shop.
ii. *Optional* - Search that is optional to it—is a parent node.
iii. *Or* - at least Bank Transfer or Credit Card must be selected.
iv. *Alternative* - High or Standard must be selected.
v. Require - if Credit Card exists, High must be selected.
vi. *Exclude* - A and B cannot exist in the same product.

Both *Require* and *Exclude* are known as Cross Tree Constraint relationships. Based on FM, test case selection process using Combinatorial Interaction Testing (CIT) will select a valid combination of features which is known as configuration.

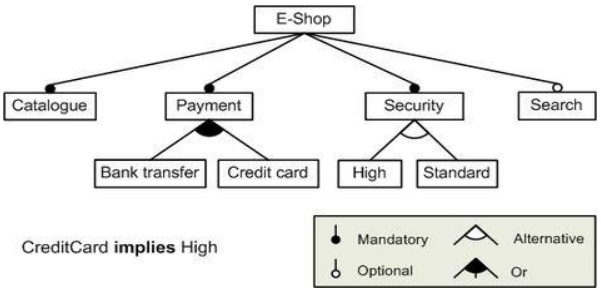

Figure 2: Feature model Electronic Shopping [10]

B. Combinatorial Interaction Testing

The challenge of SPL testing gets more complex when a large number of features are involved, which contributes to a higher number of products, thus, requiring a high testing effort. This is unfeasible in SPL testing. Thus, CIT has been proposed to reduce the number of products to a considerable amount comprising the most relevant products [6, 10]. The most relevant set of products or known as test cases is a subset of a large number of test cases. A subset which consists the most features and relationships are most welcomed as it consists the most faults to be revealed.

Existing works using CIT variation include pairwise testing (two-wise) in SPL [4, 24, 25, 26]. Pairwise testing generate possible valid combinations of features in a product. In SPL, the generation of configuration is done by using a sampling algorithm such as ICPL, Chavtal, CASA, and AETG [2, 6]. However, even after CIT has been performed, the number of test cases generated are still substantially high. Thus, TCP is recommended by many researchers to be used as it rearranges best sequence of test cases with maximum coverage to detect all faults present in a test suite.

C. Test Case Prioritization

Testing all test cases in a real SPL environment is considered as unfeasible due to limited testing time and cost. Thus, TCP is a preferable approach to be used in SPL. TCP rearranges test cases, which covers the most interaction between features to be prioritized first. TCP works under the assumption that most faults are triggered by the most different products, which consequently contributes to low testing effort and fast market release. To achieve a high rate of fault detection, a few selection criteria have been proposed by Sanchez *et al.* [10] such as Cyclomatic Complexity (CC), Cross Tree Constraint Ratio (CTCR), or a similarity-based prioritization that relies on product similarity from a test suite to determine the new order of product to be tested. The common concept of TCP is to achieve a faster rate of fault detection and a higher coverage. In this study, the authors focus on similarity criteria that have been established in existing works. SPL products will be ordered based on their similarity value ranging from zero to one, with zero denoting two products that are completely similar whereas one denoting two products that are completely different from each other.

D. Similarity Measure

The usage of similarity measures in TCP for SPL is described in this section. The authors selected six types of similarity measures, which have been considered as the best or most suitable similarity measures to be used in TCP based on existing works [3, 6, 16, 22]. The rationale of selecting six similarity measures in this study is because the authors are motivated to explore various types of similarity measures. The TCP starts by utilizing products generated by a sampling algorithm such as ICPL. The authors also show the calculation for six products and the order of products in a test suite. Further, the authors use a local maximum prioritization algorithm in order to avoid biasness. The algorithm allows the authors to determine the order of products by firstly selecting two more dissimilar products (i.e. products with the highest distance between them) and add them to a prioritization list. Secondly, the process of selecting products with the next highest distance is continued until all products have been added to the list. The list represents the order of products to be tested [1]. An example of an original order of an electronic shop's products in a test suite are shown next. The authors only selected six products and their respective calculation to save space in the paper.

Product 1 = {E-Shop, Catalogue, Payment, Bank Transfer, Security, High}
Product 2 = {E-Shop, Catalogue, Payment, Bank Transfer, Security, Standard}
Product 3 = {E-Shop, Catalogue, Payment, Credit Card, Security, High}
Product 4 = {E-Shop, Catalogue, Payment, Bank Transfer, Credit Card, Security, High}
Product 5 = {E-Shop, Catalogue, Payment, Bank Transfer, Security, High, Search}
Product 6 = {E-Shop, Catalogue, Payment, Bank Transfer, Security, Standard, Search}

1) Jaccard Distance

$$\text{Jaccard Distance} = \left(1 - \frac{|Pa \cap Pb|}{|Pa \cup Pb|}\right) \quad (1)$$

where: $p_a \cap p_b$ = Common features between Product A and B
$p_a \cup p_b$ = Total features between Product A and B

Distance among test cases using Jaccard distance is shown on Table 1 and example of calculation is shown below.

(P1 versus P2) = 1 - (5/7) = 0.286

Table 1
Jaccard distance

Product	P1	P2	P3	P4	P5	P6
P1	0.0	0.286	0.286	0.143	0.143	0.375
P2	0.286	0.0.	0.5	0.375	0.375	0.143
P3	0.286	0.5	0.0	0.143	0.375	0.444
P4	0.143	0.375	0.143	0.0	0.25	0.444
P5	0.143	0.375	0.375	0.25	0.0	0.25
P6	0.375	0.143	0.444	0.444	0.25	0.0

Order of products in test suite: P6, P3, P5, P2, P4, P1

2) Hamming Distance

$$\text{Hamming Distance } (p_a, p_b, F) = 1 - \frac{|pa \cap pb| + |(F \backslash pa) \cap (F \backslash pb)|}{|F|} \quad (2)$$

where: $p_a \cap p_b$ = Common features between Product A and B
$(F \backslash pa) \cap (F \backslash pb)$ = Features that do not exist between Product A and B
F = Total number of features in test suite

Distance among test cases using Hamming distance is shown on Table 2 and example of calculation is shown below.

(P1 versus P2) = 1 – ((5 + 2)/ 9) = 0.22

Table 2
Hamming distance

Product	P1	P2	P3	P4	P5	P6
P1	0.0	0.222	0.222	0.222	0.222	0.333
P2	0.222	0.0	0.222	0.333	0.333	0.111
P3	0.222	0.222	0.0	0.111	0.333	0.444
P4	0.222	0.333	0.111	0.0	0.222	0.333
P5	0.222	0.333	0.333	0.222	0.0	0.222
P6	0.333	0.111	0.444	0.333	0.222	0.0

Order of products in test suite: P6, P3, P5, P2, P4, P1

3) Jaro-Winkler

$$Jaro - Winkler_{(t,T-1)} = [Jaro\ (t, T_{i-1}) + \frac{p'}{10} * (1\text{-}Jaro\ (t,t)] \quad (3)$$

The first part of Jaro-Winkler is used to calculate Jaro distance, whereas, the second part of Jaro-Winkler is used as an extension to give a weight into a prefix character in the strings.

$$d_j = \frac{1}{3}\left(\frac{m}{s_1} + \frac{m}{s_2} + \frac{m-t}{m}\right) \quad (4)$$

where: m = Count of maximum number of matching characters in the same order
s_1 = Length of the first product
s_2 = Length of the second product
t = Half number of transposition of characters in strings

Whereas the second part of Winkler is given by:

$$d_{jw} = d_j + (\ell p(1\text{-}d_j)) \quad (5)$$

where: ℓ = Length of common prefix at the start of string, up to a maximum of four characters.
p = Standard weight used in Jaro-Winkler, $p=0.1$.

Distance among test cases is using Jaro-Winkler is shown on Table 3 and example of calculation is shown below.

(P1 versus P2) = 0.066
Jaro distance = 1/3 (5/6 + 5/6 + 5/5) = 0.89
Jaro-Winkler = 0.89 + ((4 x 0.1)(1-0.89)) = 0.934
0.066 = 1 – 0.934

Table 3
Jaro-Winkler

Product	P1	P2	P3	P4	P5	P6
P1	0.0	0.066	0.066	0.15	0.03	0.091
P2	0.066	0.0	0.077	0.046	0.046	0.03
P3	0.066	0.077	0.0	0.035	0.106	0.178
P4	0.15	0.046	0.035	0.0	0.091	0.115
P5	0.03	0.046	0.106	0.091	0.0	0.057
P6	0.091	0.03	0.178	0.115	0.057	0.0

Order of products in test suite: P6, P3, P5, P4, P2, P1

4) Cosine Similarity

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \quad (6)$$

where: A = Vector A
B = Vector B
||A|| = Magnitude of vector A
||B|| = Magnitude of vector B

There are two other formula used to calculate Vector (Dot Product) and Magnitude of product:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \quad (7)$$

$$\|\mathbf{x}\| := \sqrt{x_1^2 + \cdots + x_n^2}. \quad (8)$$

The authors use Term Frequency (TF) to calculate the number of occurrences of features inside a product. Typically, TF value will be available as a Frequency of Occurrence Vector (FOV). However, in SPL, features will not generate two or more times from a similar product. Thus, the TF is modified to a Binary Occurrence of Vector (BOV). An example of the difference between Binary and Frequency using "Apple" is shown as follows.

Binary Occurrence Vector of "apple"
1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,**1**,0,0,0,0,0,0,0,0,0,0

Frequency of Occurrence Vector of "apple" 1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,**2**,0,0,0,0,0,0,0,0,0,0,0

Product features:
BOV :

a	**b**	**c**	**d**	**e**	**f**	**g**	**h**
1	1	1	1	1	1	1	1

Distance among test cases is using Cosine Similarity is shown on Table 4 and example of calculation is shown below.

(P1 versus P2) = 0.167

Dot Product = 5
Magnitude of Product A = √6
Magnitude of Product B = √6
Magnitude of product A and B = √6 * √6 = 6
Dot Product / product of magnitude of A and B = 5/6 = 0.8333
1 – 0.8333 = 0.167

Table 4
Cosine Similarity

Product	P1	P2	P3	P4	P5	P6
P1	0.0	0.167	0.167	0.074	0.074	0.228
P2	0.167	0.0	0.333	0.228	0.228	0.074
P3	0.167	0.333	0.0	0.074	0.074	0.383
P4	0.074	0.228	0.074	0.0	0.143	0.283
P5	0.074	0.228	0.074	0.143	0.0	0.143
P6	0.228	0.074	0.383	0.283	0.143	0.0

Order of products in test suite: P6, P3, P2, P4, P5, P1

5) Counting function

$$(Pa, Pb) = 1 - \frac{c}{((\ell a + \ell b)/2)} \quad (9)$$

where: c = Common features of Product A and B
ℓa= Length of product A
ℓb = Length of product B

Distance among test cases is using Counting Function is shown on Table 5 and example of calculation is shown below.

(P1 versus P2) = 0.167

Counting function = 1 – (5 / ((6+6)/2))
= 0.167

Table 5
Counting Function

Product	P1	P2	P3	P4	P5	P6
P1	0.0	0.167	0.167	0.077	0.077	0.231
P2	0.167	0.0	0.333	0.231	0.231	0.077
P3	0.167	0.333	0.0	0.077	0.231	0.385
P4	0.077	0.231	0.077	0.0	0.142	0.286
P5	0.077	0.231	0.231	0.142	0.0	0.286
P6	0.231	0.077	0.385	0.286	0.286	0.0

Order of products in test suite: P6, P3, P2, P4, P5, P1

6) Sorensein Similarity

$$(Pa, Pb) = 1 - \frac{2c}{\ell a + \ell b} \quad (10)$$

where: c = Common features of Product A and B
ℓa= Length of product A
ℓb = Length of product B

Distance among test cases is using Sorensein Similarity is shown on Table 6 and example of calculation is shown below.

(P1 versus P2) = 0.167

Sorensein = 1 – (2(5) / (6+6))
= 0.167

Table 6
Sorensein Similarity

Product	P1	P2	P3	P4	P5	P6
P1	0.0	0.167	0.167	0.077	0.077	0.231
P2	0.167	0.0	0.333	0.231	0.231	0.077
P3	0.167	0.333	0.0	0.077	0.231	0.385
P4	0.077	0.231	0.077	0.0	0.142	0.286
P5	0.077	0.231	0.231	0.142	0.0	0.286
P6	0.231	0.077	0.385	0.286	0.286	0.0

Order of products in test suite: P6, P3, P2, P4, P5, P1

IV. EXPERIMENT AND RESULTS

In this section, the authors evaluate six similarity measures described in section III by comparing their effectiveness towards existing SPL feature models by using sampling algorithm ICPL (T=2), since it is the fastest algorithm available in the market (6). The objective is to conduct an experimental study on similarity measures on TCP for SPL. In order to achieve this objective, the authors provide three research questions as follows:

RQ1: What is the best similarity measure to be used in TCP in terms of rate of fault detection?
RQ2: Is the ordering of features based on similarity measures inside SPL product contributes to a better result?
RQ3: What factors contribute to a better result for TCP based on similarity measures?

A. Experimental Settings

In order to evaluate the comparison of similarity measures highlighted in section III, the authors used prototype developed by Sanchez *et al.* [10]. The prototype provides an integration of existing tools such as SPLCAT tool for generation of configuration and prioritization process. The authors utilized six similarity measures and evaluated each of the similarity measure on existing feature models on SPLOT repository. The experimentation was performed using Windows 8.1 equipped with Intel Core I5-3337U 1.8GHz with 6GB of RAM.

1) Feature Models

The authors implemented their similarity measures on real feature models taken from SPLOT repository that houses a wide range of features, from 40 to 300 features. The feature range was selected as some of benchmarked feature models within the range have been used in existing works [6], [10]. Based on Table 7, the authors classified feature models into three groups. Small for feature models of below than 50 features, while medium for feature models between 50 to 100 features, and large for feature models of more than 100

features. Details such as generated product, CTCR, and number of faults used in experiment are shown below:

Table 7
Feature Models details

Feature Model	Features	Generated products	CTCR	Faults
Web portal	43	19	25%	4
Video Player	71	18	0%	4
Car Selection	72	24	31%	4
Go Phone	77	14	14%	4
Model Transformation	88	28	0%	8
Battle of tanks	144	484	0%	12
Printers	172	129	0%	16
Electronic Shopping	290	24	11%	28

2) Fault generation and evaluation metric

This work utilized a fault simulator as appeared in Bagueri *et al.* [31], comprising two to four features of interaction fault. The fault simulator has been used in existing SPL works to investigate the effectiveness of TCP technique [6], [10]. Two to four features' interactions are chosen due to practical application in a real SPL testing [31]. Typically, the general assumption of faults is that faults are distributed equally among features in a product, however, some researchers have argued that faults may be detected where they have not been unexpected [6]. Despite this, the best way to test SPL is by sticking to the assumption that there are equally distributed faults in features compared to focusing on certain features.

3) The authors utilized Average Percentage of Fault Detected (APFD) as an evaluation metric in order to investigate the rate of faults detected by similarity measures [27].

APFD metric evaluates the effectiveness of prioritization by calculating the average number of faults exposed based on their index position in a prioritized test suite. A higher APFD indicates a more effective similarity measure in detecting faults. The equation of APFD metric is as follows:

$$APFD = 1 - \frac{TF_1+TF_2+\ldots+TF_n}{n\times m} + \frac{1}{2n} \quad (11)$$

where T = test suite
n = Test cases
TFi = Position of the first test case exposing fault
m = Number of faults exposed by test suite

Table 8
Test suite and faults detected

Test/Faults	F1	F2	F3	F4	F5
T1	x	x			
T2	x		x		
T3	x	x	x	x	
T4					x

Example of APFD calculation:

= (1 - (1 + 1 + 1 + 1 + 3) / 4 x 5) + 1/(2x4)
= 0.78 (Based on ordering T3, T2, T4, T1)

B. Experimental Setup

Based on Figure 3, the authors operate four phases in their experiment. First, four-wise interaction faults are generated based on feature models selected using fault generator. Second, valid combination of features are generated using SPLCAT tool with sampling algorithm ICPL (T=2). Third, similarity of generated products are calculated using six similarity measures, and subsequently prioritized according to their distance with product having the highest distance will be placed on top of prioritization list. Any value obtained that is not between zero and one is considered as an error from the algorithm. Fourth, prioritized products are evaluated by fault simulation and rate of faults detected is calculated by using APFD. The authors iterate the steps for ten times to gain a balanced result.

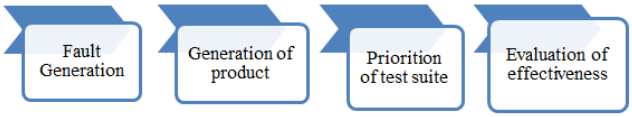

Figure 3: Phases of evaluation experiment

V. Discussion

The result of evaluation is shown in Table 9. The average result of APFD for all eight feature models ranges from 79% to 85%. Jaro-Winkler has the highest average APFD value of 84.96%, followed by Cosine (84.32%), third, Counting function (83.73%), fourth, Sorensein (83.73%), fifth, Hamming distance (82.69%), and lastly, Jaccard distance (79.4%). Jaro-Winkler achieved the highest APFD value across seven feature models. Other than that, across three feature models, it achieved similar score as Hamming distance and Cosine similarity as shown in bold in Table 9. The results help the authors to answer RQ1, whereby Jaro-Winkler is the best similarity measure to be used to increase the rate of fault detection of proposed TCP for SPL technique. On the other hand, Jaccard distance scored much lower APFD value on most feature models compared to the rest of similarity measures. Despite that, Jaccard distance achieved the highest APFD value for model transformation feature model. From the results, a small feature model tends to have similar APFD result across different similarity measures. This is because small feature model typically produces less diversity of test cases.

Furthermore, Sorensein and Counting function similarity measures achieved identical results across all feature models. This occurred as they comprise similar formula, but with different ways of calculating similarity. This problem can be solved by adjusting the current similarity measure formula to fit SPL domain. For example, commonalities and variabilities concepts can be adjusted using Winkler formula from Jaro-Winkler. Winkler extension increases the weight of prefix in strings, which is represented by common features of a product in SPL. By doing this, similarity value will vary, which will provide an accurate result and easier ranking process of products. The formula greatly benefits testers in order to diversify similar distance values among test cases. Current technique will choose the first product they discover in determining the ranking of a product. The flexibility of the Jaro-Winkler formula is suitable for SPL concept, which will allow it to handle commonalities and variabilities in a systematic approach. This answers the second RQ2. Finally, based on the Figure 4. Jaro-Winkler achieved the steepest curve and detected 100% of faults much earlier than other similarity measures, utilizing only 70% of test suite for

Electronic Shopping compared to other similarity measures. This shows that the rate of fault detection of Jaro-Winkler is the best. Moreover, Jaro-Winkler was implemented across all eight feature models—from small to large feature models, available from SPLOT repository.

Table 9
APFD Result for eight feature models

Feature Model	Test Suite Size	No. of features	Faults Detected	APFD					
				Jaccard Distance	Hamming Distance	Jaro-Winkler	Cosine Similarity	Counting Function	Sorensein
Web portal	19	43	4/4	85.83	**95.83**	**95.83**	**95.83**	92.5	92.5
Video Player	18	71	4/4	89.17	**94.17**	**94.17**	**94.17**	**94.17**	**94.17**
Car Selection	24	72	4/4	49	**54**	**54**	52	52	52
Go Phone	14	77	4/4	75	75	**91.6**	**91.6**	87.5	87.5
Model Transformation	28	88	7/8	**77.01**	76.43	75.86	74	**77.01**	**77.01**
Battle of tanks	484	144	11/12	83.6	83.78	**84.81**	84.46	84.46	84.46
Printers	129	172	15/16	95.39	95.43	**95.76**	95.62	94.86	94.86
Electronic Shopping	24	290	25/28	84.23	86.84	**87.61**	86.86	87.3	87.3
Average APFD				79.90	82.69	**84.96**	84.32	83.73	83.73

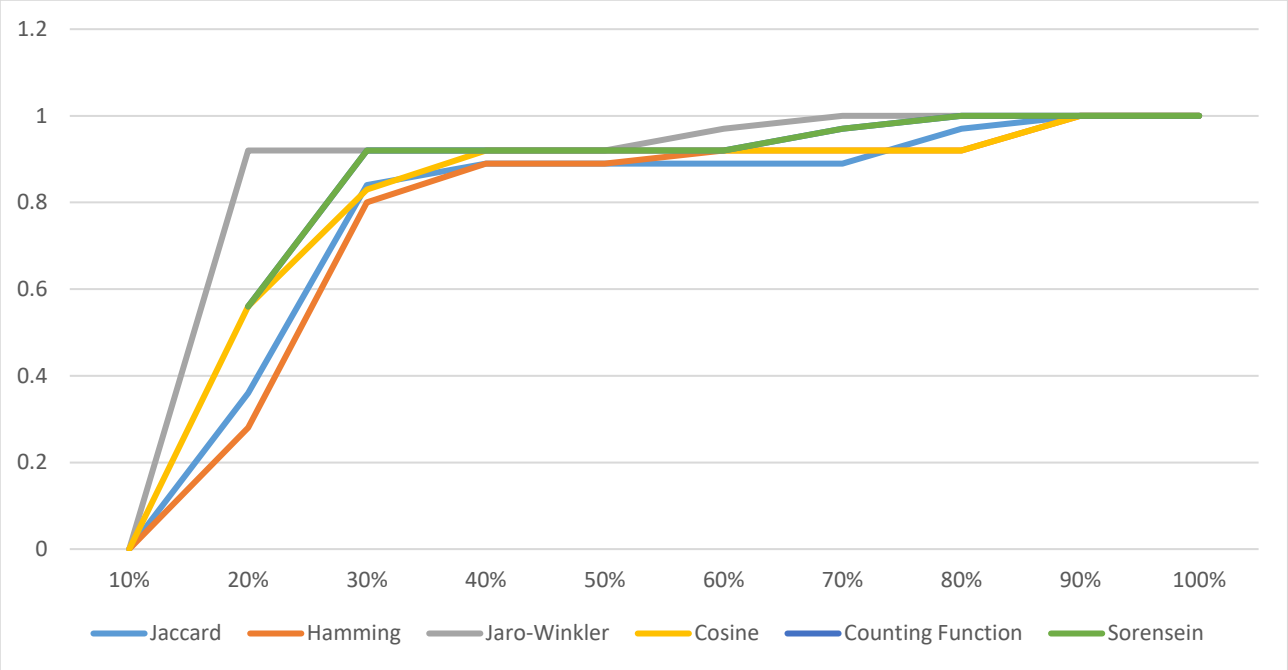

Figure 4: APFD metrics for Electronic Shopping

Next, for RQ3, among the factors that contribute to a better result based on similarity measure is how the product was generated through sampling algorithm. For example, ICPL may produce randomized products with valid combination of features. This affects calculation of similarity measures especially for similarity formula that considers the placement of features. Another factor is the generation of faults, whereby faults generated by fault simulator will yield different result towards APFD value. Thus, realizing the importance of real faults in case studies will greatly contribute to a much better testing process.

VI. Threat to Validity

In this section, the authors deliberate potential threats to current study. Earlier, the authors did mention that this work applied local maximum prioritization to prioritize test suite. The algorithm selects two products with the furthest distance between them since it is assumed to contain the most different feature between each other. Each TCP technique has its own algorithm to help in determining prioritization ranking in order to contribute to a much better prioritization process [6]. In this study, a local maximum prioritization was chosen as it has been used by many researchers in TCP field [3],[10].

Moreover, not all of the faults were detected by similarity measures. This is due to the absence of real faults data which are required. Fault generator utilized in the experimentation simulated error based on the interaction of features inside a product. This problem has been highlighted by many researchers, attributing it to a lack of real data with real faults [6], [10]. However, the current work aimed to investigate the rate of faults detection by similarity measures, not the number of faults detected. Additionally, the assumption of equally distributed faults is much better for testing process compared to non-idealized idea without facts. Another threat to validity is an uncertainty nature of SPL testing that produces a different result after each run. However, this problem was solved in the experiment by executing multiple runs.

VII. Conclusion and Future Work

As a conclusion, this study provides a comparison of similarity measures across eight benchmark case studies used by other researchers. The work extends the prototype

developed by Sanchez *et al*. [10]. The results show that Jaro-Winkler achieved the best result as a similarity measure. This opens up improvement opportunities through similarity measure's flexibility of formula to suit various domains especially SPL's general concept in commonalities and variabilities. This will subsequently influence testing process in terms of the ordering of products and features within a test suite.

For future work, the authors are determined to extend the experiment on real industrial case studies with real faults and with similar similarity measures. Furthermore, this study is capable of increasing maximum fault interactions to six features interactions since it will detect almost all faults [30]. The authors are also determined to explore the extension of prioritization algorithm in order to enhance the ranking process to be more efficient in terms of execution time. Moreover, the authors will explore further on various combination of existing similarity measures extensively such as hybrid similarity measures in order to produce improved APFD result as suggested by [38].

ACKNOWLEDGEMENTS

We are grateful for UTM scholarship to Author 1. Furthermore, we express our gratitude profoundly to Research University Grant (GUP), Universiti Teknologi Malaysia (UTM) under Cost Centre No Q.J130000.2528.15H44, for their financial support. Our profound appreciation also goes to ERetSEL lab members for their continuous support towards the completion of this paper.

REFERENCES

[1] X. Davier, G. Perrouin, M. Cordy, P. Schobbens, A. Legay, and P. Heymans, "Towards statistical prioritization for software product lines testing," in *Proc. of the Eighth Int. Workshop on Variability Modelling of Software-Intensive Systems*, 2014, pp. 10.

[2] A. Ensan, E. Bagheri, M. Asadi, D. Gasevic, and Y. Biletskiy, "Goal-oriented test case selection and prioritization for product line feature models," in *IEEE Eighth Int. Conf. on Information Technology: New Generations (ITNG), 2011*, 2011, pp. 291-298.

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

[4] M. Johansen, Ø. Haugen, F. Fleurey, A. Eldegard, and T. Syversen. "Generating better partial covering arrays by modeling weights on sub-product lines," in *Model Driven Engineering Languages and Systems*, R. B. France, J. Kazmeier, R. Breu, and C. Atkinson, Eds. Berlin, Heidelberg: Springer, 2012, pp. 269-284.

[5] R. Lachmann, S, Lity, S. Lischke, S. Beddig, S. Schulze, and I. Schaefer, "Delta-oriented test case prioritization for integration testing of software product lines," in *Proc. of the 19th Int. Conf. on Software Product Line*, 2015, pp. 81-90.

[6] M. Al-Hajjaji, T. Thüm, J. Meinicke, M. Lochau, and G. Saake, "Similarity-based prioritization in software product-line testing," in *Proc. of the 18th ACM International Software Product Line Conference-Volume 1*, 2014, pp. 197-206.

[7] Pohl, Klaus, and A. Metzger. "Software product line testing," *Communications of the ACM*, vol. 49, no. 12, pp. 78-81, 2006.

[8] P. Bielkowicz, P. Patel, and T. T. Tun, "Evaluating information systems development methods: a new framework," in *Int. Conf. on Object-Oriented Information Systems*, 2002, pp. 311-322.

[9] J. Lee, S. Kang, and D. Lee, "A survey on software product line testing," in *Proc. of the ACM 16th Int. Software Product Line Conference-Volume 1*, 2012, pp. 31-40.

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

[11] S. Yoo, and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67-120, 2012.

[12] A. Ahnassay, E. Bagheri, and D. Gasevic, *Empirical Evaluation in Software Product Line Engineering*. Technical report, Tech. Rep. TR-LS3-130084R4T, Laboratory for Systems, Software and Semantics, Ryerson University, 2013.

[13] E. Engström, and P. Runeson, "Software product line testing–a systematic mapping study," *Information and Software Technology*, vol. 53, no. 1, pp. 2-13, 2011,

[14] P. A. M. S. Neto, I. C. Machado, J. D. McGregor, E. S. Almeida, and S. R. L. Meira. "A systematic mapping study of software product lines testing," *Information and Software Technology*, vol. 53, no. 5, pp. 407-423. 2011.

[15] I. C. Machado, J. D. Mcgregor, and E.S. Almeida. "On strategies for testing software product lines: A systematic literature review," *Information and Software Technology*, 2014, vol. 56, no. 10 pp.1183-1199, 2014.

[16] B. P. Lamancha, M. Polo, and M. Piattini, "Systematic review on software product line testing," in *Int. Conf. on Software and Data Technologies*, 2010, pp. 58-71.

[17] I. C. Machado, J. D. McGregor, and E. S. Almeida. "Strategies for testing products in software product lines," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 6, pp. 1-8, 2012.

[18] L. Jin-Hua, L. Qiong, and L. Jing. "The w-model for testing software product lines," in *Int. Symposium on Computer Science and Computational Technology (ISCSCT'08)*, 2008, pp. 690-693.

[19] G. J. Myers, C. Sandler, and T. Badgett. *The Art of Software Testing*. John Wiley & Sons, 2011.

[20] T. Thomas, S. Apel, C. Kästner, I. Schaefer, and G. Saake. "A classification and survey of analysis strategies for software product lines," *ACM Computing Surveys (CSUR)*, vol. 47, no. 1, p. 6, 2014.

[21] R. E. Lopez-Herrejon, S. Fischer, R. Ramler, and A. Egyed, "A first systematic mapping study on combinatorial interaction testing for software product lines," in *2015 IEEE Eighth Int. Conf. on Software Testing, Verification and Validation Workshops (ICSTW)*, 2015, pp. 1-10.

[22] S.S. Choi, S. H. Cha, and Charles C. Tappert. "A survey of binary similarity and distance measures," *Journal of Systemics, Cybernetics and Informatics*, vol. 8, no. 1, pp.43-48, 2010.

[23] M. Bilenko, "Learnable similarity functions and their applications to clustering and record linkage," in *AAAI'04 Proceedings of the 19th National Conference on Artificial Intelligence*, 2004, pp. 981-982.

[24] S. Oster, F. Markert, and P. Ritter. "Automated incremental pairwise testing of software product lines," in *Software Product Lines: Going Beyond*, J. Bosch, and J. Lee, Eds. Berlin, Heidelberg: Springer, 2010, pp.196-210.

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

[26] G. Perrouin,, S. Sen, J. Klein, B. Baudry, and Y. L. Traon, "Automated and scalable t-wise test case generation strategies for software product lines," in *2010 Third Int. Conf. on Software Testing, Verification and Validation (ICST)*, 2010, pp. 459-468.

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

[28] X. Devroey, G. Perrouin, A. Legay, P. Schobbens, and P. Heymans, "Search-based similarity-driven behavioural SPL testing," in *Proc. of the Tenth International Workshop on Variability Modelling of Software-intensive Systems*, 2016, pp. 89-96.

[29] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. Spencer Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. No. CMU/SEI-90-TR-21. Carnegie-Mellon Univ. Pittsburgh Pa Software Engineering Inst, 1990.

[30] D. R. Kuhn, D. R. Wallace, and A. M. Gallo. "Software fault interactions and implications for software testing," *IEEE Trans. on Software Engineering*, vol. 30, no. 6, pp.418-421, 2004.

[31] F. Ensan, E. Bagheri, and D. Gasevic, "Evolutionary search-based test generation for software product line feature models," in *CAiSE*, vol. 7328, 2012, pp. 613-628.

[32] M. Muja, and DG. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2227-2240, 2014.

[33] M. Kowal, S. Schulze, and I. Schaefer. "Towards efficient SPL testing by variant reduction," in *Proc. of the 4th international workshop on Variability & composition*, 2013, pp. 1-6.

[34] H. Hemmati, and L. Briand, "An industrial investigation of similarity measures for model-based test case selection," in *2010 IEEE 21st Int.*

Symposium on Software Reliability Engineering (ISSRE), 2010, pp. 141-150.

[35] 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, 2014.

[36] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran. "Prioritizing test cases with string distances," *Automated Software Engineering*, vol. 19, no. 1, pp.65-95, 2012

[37] D. Hao, L. Zhang, L. Zang, Y. Wang, X. Wu, and T. Xie, "To be optimal or not in test-case prioritization," *IEEE Trans. on Software Engineering*, vol. 42, no. 5 pp. 490-505, 2016.

[38] W. Cohen, P. Ravikumar, and S. Fienberg. "A comparison of string metrics for matching names and records," in *KDD workshop on Data Cleaning and Object Consolidation*, 2003, pp. 73-78.