How to cite this paper:

Halim. S., Jawawi, D. N., & Sahak. M. (2018). Similarity distance measure and prioritization algorithm for test case prioritization in software product line testing. *Journal of Information and Communication Technology, 18*(1), 57-75.

SIMILARITY DISTANCE MEASURE AND PRIORITIZATION ALGORITHM FOR TEST CASE PRIORITIZATION IN SOFTWARE PRODUCT LINE TESTING

Shahliza Abd Halim, Dayang Norhayati Abang Jawawi & Muhammad Sahak
School of Computing, Faculty of Engineering
Universiti Teknologi Malaysia, Malaysia

shahliza@utm.my; dayang@utm.my; muhammadbsahak@gmail.com

ABSTRACT

To achieve the goal of creating products for a specific market segment, implementation of Software Product Line (SPL) is required to fulfill specific needs of customers by managing a set of common features and exploiting the variabilities between the products. Testing product-by-product is not feasible in SPL due to the combinatorial explosion of product number, thus, Test Case Prioritization (TCP) is needed to select a few test cases which could yield high number of faults. Among the most promising TCP techniques is similarity-based TCP technique which consists of similarity distance measure and prioritization algorithm. The goal of this paper is to propose an enhanced string distance and prioritization algorithm which could reorder the test cases resulting to higher rate of fault detection. Comparative study has been done between different string distance measures and prioritization algorithms to select the best techniques for similarity-based test case prioritization. Identified enhancements have been implemented to both techniques for a better adoption of prioritizing SPL test cases. Experiment has been done in order to identify the effectiveness of enhancements done for combination of both techniques. Result shows the effectiveness of the

Received: 15 July 2018 Accepted: 3 September2018 Published: 11 December 2018

combination where it achieved highest average fault detection rate, attained fastest execution time for highest number of test cases and accomplished 41.25% average rate of fault detection. The result proves that the combination of both techniques improve SPL testing effectiveness compared to other existing techniques.

Keywords: Combinatorial interaction testing, similarity distance, string based prioritization, feature model, sampling algorithm.

INTRODUCTION

Software Product Line (SPL) engineering is based on systematically managing and exploiting the commonalities and variabilities between features to achieve specific goals of customers (Al-Hajjaji, Lity, Lachmann, Thüm, Schaefer., & Saake, 2016). The adoption of SPL in many software organizations is to exploit the advantages of reducing development cost, time and effort while preserving the quality of products. Feature Model (FM) is used in SPL to provide detailed information regarding to features and relationship between the features. Furthermore, the commonalities and variabilities of all products also can be identified from the FM. These features will undergo configuration process to select only a set of valid combination of the features known as configuration. Quality assurances such as testing in SPL is much harder compared to single system due to complexity of features from the FM, thus new technique is suggested to be created for this challenge such as reusing test assets to improve efficiency and handle complexity of SPL (Johansen et al., 2012). Although product-by-product testing can be done in SPL, it is highly infeasible in terms of cost and time thus incremental testing strategy is preferable to be used in SPL (Al-Hajjaji et al., 2016). However, combinatorial explosion further complicates SPL testing due to the number of products that increases exponentially when the number of features grows. To overcome combinatorial explosion of products, regression testing strategy such as Test Case Prioritization (TCP) is preferred to be used in SPL since TCP is able to reduce the testing resources allocated while preserving the number of test cases and maintaining efficient fault detection. TCP is one of the regression testing strategies along with minimization and selection proposed by a survey in Machado et al. (2014). In SPL, various researchers proposed TCP into their works (Devroy, Perrouin, Cordy, Samih, Legay, Schobbens & Heymans, 2017; Henard, Papadakis, Perrouin, Klein, Heymans & Le Traon, 2014; Al-Hajjaji et al., 2017; Johansen, Haugen, Fleurey, Eldegard., & Syversen, 2012) to overcome issues such as combinatorial explosion. Among the most promising approach for TCP is a similarity-based prioritization technique

used by Henard et al. (2014) and Al-Hajjaji et al. (2016) to effectively test SPL products. The aim of similarity-based prioritization is to reorder the test cases in the prioritized test suite. This is done to increase this method's capability to detect faults earlier in the suite thus significantly reduce the testing resources allocated and accelerate the time to market the product. Similarity-based prioritization acts upon the assumption of the most dissimilar test cases are able to detect high number of faults compared to similar ones (Henard et al., 2014). Typically, similarity-based prioritization techniques consist of two important key elements which are similarity measures to calculate the similarity between test cases and prioritization algorithm to reorder the test cases in the test suite according to their similarity value. Thus, we are motivated to compare between different similarity distance measures and prioritization algorithms in order to find the best combination of both techniques which can be further enhanced to fulfill the goal of increasing the probability of finding faults in test cases. An experiment will be done to evaluate the effectiveness of the proposed combination. Therefore there are three contributions of this study as follows. The first is comparing between different similarity distance measures and prioritization algorithms and to identify the best techniques for similarity-based prioritization. Enhancing both type of techniques and produce a better method for adoption in SPL testing is the second contribution. The third contribution is performing an experiment to evaluate the effectiveness of the integration of both enhanced techniques and identifying further improvement (if any) compared to the existing techniques.

RELATED WORKS

Combinatorial Interaction Testing (CIT) is commonly used in SPL testing to produce a set of products based on the identification of relationship between the features from FM. CIT is able to reduce the number of products generated compared to the very high number of products to be tested using product-by-product testing (Al-Hajjaji et al., 2017). Several CIT sampling algorithms have been proposed to overcome combinatorial explosion such as AETG, CASA, ICPL, Chvatal, and MoSo-PoLiTe (Cohen, Ravikumar, & Fienberg., 2008; Garvin, Cohen & Dwyer, 2011; Johansen Haugen, Fleurey, Eldegard & Syversen, 2012; Chvatal, 1979; Oster, Zorcic, Markert & Lochau , 2011). Among the highly regarded sampling algorithm is ICPL by Johansen et al. (2012) that is proposed to tackle scalability problem by producing acceptable size of covering arrays. This method also has faster execution time which is important in SPL testing. However, there are still large number of test cases need to be tested thus prioritization is needed to rank test cases from the test suite to enable a high probability

of faults detected at an earlier rate (Henard et al., 2014). Several TCP criteria have been proposed by Sanchez et al. (2014) such as Cross Tree Constraint Ratio (CTCR), Coefficient of Connectivity-Density (CoC), Variability Coverage and Cyclomatic Complexity (VC & CC) and also Commonality and Dissimilarity based on the features and their relationship from the FM. These criteria are used to prioritize the test cases to achieve maximum capability to detect faults from the test cases since TCP only need subsets of test cases from test suite to detect faults. Moreover, several TCP works also consider different issues such as behavior of the features from the products (Devroey, Perrouin, Cordy, Samih, Legay, Schobbens and Heymans, 2017; Zamli, Klaib, Younis & Yeh, 2010) and prioritization of test cases in the integration level of testing for SPL (Al-Hajjaji et al., 2017; Devroy et al., 2014). Criteria such as dissimilarity used by Henard et al. (2014) and Al-Hajjaji et al. (2016) is highly preferable. Dissimilarity prioritization consists of two elements namely similarity measures to calculate similarity/dissimilarity value from test cases and prioritization algorithm to rank the order of test cases. These two elements are the main focus of this work which is also used by several other researchers (Henard et al., 2014; Al-Hajjaji et al., 2017; Sanchez, Seguira & Ruiz-Cortis, 2014).

Recently, various types of TCP technique have been proposed in SPL to overcome issue of combinatorial explosion caused by variabilities of features. Recently, statistical prioritization (Devroey et al., 2017) is proposed for SPL since most existing TCP techniques do not consider behavior of products and depend solely on FM. This technique uses usage models with Markov chains for prioritizing behavior based on the usage of the products The work suggested a new way for reuse of test assets based on behavior or scenarios of the product. Another work (Al-Hajjaji et al., 2017) implemented delta modeling into similarity-based prioritization for solution-space approach to improve effectiveness of SPL testing. Their work found that SPL testing effectiveness can be improved by incorporating delta modeling into similarity-based prioritization. Meanwhile for similarity-based prioritization, work by Henard et al. (2014) tried to solve scalability issue by using similarity heuristic and search-based approach on large feature models. The work found that the most dissimilar test cases in the test suite are able to increase the rate of fault detection compared to similar ones which significantly increase the effectiveness of the technique. Another work by Al-Hajjaji et al. (2016) investigated random order, interaction-based approach and default order of test case by proposing new approach to incrementally select the most dissimilar test cases in the prioritized test suite to increase effectiveness of the technique. Various researchers have applied similarity measures in SPL and among the techniques frequently used to calculate similarity value among test cases is Jaccard

distance (Henard et al., 2014; Al-Hajjaji et al., 2016; Sanchez, Seguira & Ruiz-Cortis, 2014). Jaccard distance is regarded as the most efficient similarity measure to be used in SPL along with Hamming distance based on work by Devroey, Perrouin, Legay, Schobbens and Heymans (2016) that compared similarity measures such as Hamming distance, Jaccard distance, dice, anti-dice and Levenstein. Meanwhile, work by Al-Hajjaji et al. (2017) proposed enhancement on Hamming distance by considering deselected features into the formula in order to accurately calculate similarity value between test cases. Their work showed a promising result to increase the effectiveness of similarity-based prioritization techniques.

The use of similarity measures to calculate differences between two test cases can be improved using prioritization algorithms to reorder the test cases and increase the capability of fault detection. Several prioritization algorithms have been proposed in SPL such as Local Maximum, Global Maximum & All-yes config (Henard et al., 2014; Al-Hajjaji et al., 2017; Sanchez, Seguira and Ruiz-Cortis, 2014). Each algorithm ranks the test cases differently according to its aim, for example Local Maximum focuses on selecting two test cases with the highest distance between each other, while Global Maximum integrates Local Maximum distance to rank the first two test cases. However, the algorithm differentiates their process after that by selecting next test case with the highest distance to all test cases already selected on the prioritized test suite. Meanwhile, All-yes config selects test case with the most features as the first test case in the prioritized test suite. Based on the work we analysed, to the best of our knowledge, there is no extensive comparison done on existing works for both similarity distance and prioritization algorithm topics.

SIMILARITY-BASED PRIORITIZATION TECHNIQUE

This section describes concepts related to the above topic which comprise of FM as the main model for the input to generate test cases. Consequently, similarity measure comparison and enhancements done on the best similarity distance measure will be explained. Lastly, this section will explain on comparison and enhancements done to improve the existing prioritization algorithm.

Feature Model

FM is typically used in SPL to describe features and relationships between each feature which is introduced by Kang et al. (1990) in the Feature-Oriented Domain Analysis (FODA). Figure 1 is an example of Electronic Shopping FM which consists of features and their relationships.

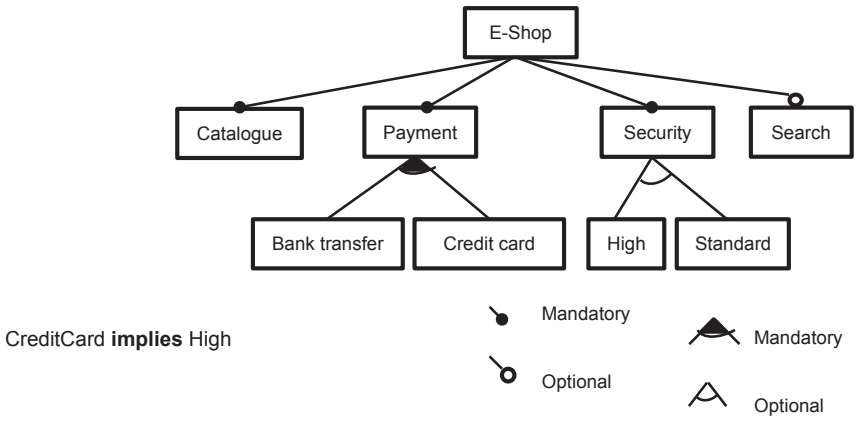

Figure 1. Feature Model for Electronic Shopping.

Among the relationships that exist in this FM is: i) mandatory where a feature is mandatory to parent node, ii) optional where a feature is optional to parent node, iii) or where at least one of the features must be selected, iv) alternative where features in the child node must be selected, v) require where both features must be existing in the same product, and vi) exclude where both features cannot exist in the same product. ICPL is used as the sampling algorithm and the sample configurations generated from this tool is as shown in Table 1.

Table 1

Sample of Test Cases Generated from Sampling Algorithm

Test Case	**Test Case Content**
T1	{E-Shop, Catalogue, Payment, Bank Transfer, Security, High}
T2	{E-Shop, Catalogue, Payment, Bank Transfer, Security, Standard}
T3	{E-Shop, Catalogue, Payment, Credit Card, Security, High}
T4	{E-Shop, Catalogue, Payment, Bank Transfer, Credit Card, Security, High}
T5	{E-Shop, Catalogue, Payment, Bank Transfer, Security, High, Search}
T6	{E-Shop, Catalogue, Payment, Bank Transfer, Security, Standard, Search}

Enhanced Similarity Distance Measure

Based on our previous work, we compared six similarity measures such as Jaccard distance, Hamming distance, Cosine Similarity, Counting function, Sorensein Similarity and Jaro-Winkler. The result shows Jaro-Winkler is the best string distance for eight feature models used (Sahak, Jawawi & Halim, 2017). The work discussed the benefits of Jaro-Winkler similarity measure in calculating the similarity value of test cases such as the usage of weight towards prefix in the strings which is common features of the products in SPL. For similarity-based prioritization, most of the existing works (Henard et al., 2014; Al-Hajjaji et al., 2017; Sanchez, Seguira & Ruiz-Cortis, 2014) used Jaccard distance and Hamming distance similarity measures since both similarity measures are regarded as the most efficient string distance to be used for SPL (Devroy et al., 2016).

This study proposes an enhanced string distance based on hybridization of Jaro-Winker and Hamming Distance equation. The purpose of this enhancement is to increase the effectiveness of similarity-based prioritization technique. In the proposed enhanced similarity measures, we want the similarity measures to accurately calculate the similarity value between test cases and to be more diverse in terms of similarity value since the more diverse the value, the easier for prioritization algorithm to determine the ranking of test cases. This is important since we want to avoid 50-50 situation in our ranking process. 50-50 situation is the situation where there are two or more test cases which have the same similarity value. Thus, prioritization algorithm is required to select between these same values as the next test case in the prioritized test suite. Typically, prioritization algorithms will select the first test case they found with the same value which sometimes is not the best test case to be chosen.

Hybridization of Jaro-Winkler and Hamming distance equations considers the deselected features from Hamming distance combined into the existing Jaro distance equation, D*j*. This combination is made due to the assumption of faults are often found in unexpected places especially in real practice (Al-Hajjaji et al., 2016). After the deselected features are implemented, this study proposes a new usage of Degree of Difference, *Df* to calculate the difference between two test cases using their length of string in order to produce their difference value. Tumeng (2017) used *Df* to replace transposed character since the original equation is used only on record linkage which is a different domain compared to SPL with features because SPL does not repeat certain feature in their test case. For example, in first name string, "A" character is repeated 2 times in "ahmad" but in SPL one feature can only be used once per test case since there is no repeated feature in a test case. This modification is also motivated by the suggestion of Choi, Szakal, Chen, Branzei, and Zhao (2010) to modify existing string distance to suit different domains such as SPL. The enhanced Jaro-Winkler string distance is as shown in equation (1).

$$d_{ej} + d_{ejw}(T_1, T_2) = \frac{1}{3}\left(\frac{m}{T_1} + \frac{m}{T_2} + \frac{n - D_f}{n}\right) + (D_f(1 - d_j)) \quad (1)$$

where

d_{ejw}	new enhanced Jaro-Winkler based on Jaro-Winkler with addition of deselected features and Degree of Difference, Df.
D_f	"degree of difference" measured using equation, $\frac{\lvert T_i - T_i + 1\rvert}{\lvert T_i - T_i + 1\rvert}$ where, $\lvert T_i - T_i + 1\rvert$ is the absolute value of test case *i* minus a subsequent test case *i+1*. Meanwhile, $\lvert T_i - T_i + 1\rvert$ is the total of length of test case i and test case *i+1*.
n	number of deselected features in both test cases.
m	Is the count of maximum number of matching characters between test case 1 and 2
T_1	Is the length of test case 1
T_2	Is the length of test case 2

Distances among test cases are calculated using the configuration of Table 1, we use two test cases T1 and T4 as follows;

T1 = {E-Shop, Catalogue, Payment, Bank Transfer, Security, High}

T4 = {E-Shop, Catalogue, Payment, Bank Transfer, Credit Card, Security, High}

Common features of both test cases $m = 6$; length of T1 and T4 respectively is 6 and 7; their degree of difference are as follows

$$D_f(\text{T1,T1}) = \frac{|T1 - T1|}{T1 + T1} = \frac{|6-6|}{6+6} = \frac{0}{12} = 0$$

$$D_f(\text{T1,T4}) = \frac{|T1 - T4|}{T1 + T4} = \frac{|6-7|}{6+7} = \frac{1}{13} = 0.0769$$

$$D_f(\text{T4,T1}) = \frac{|T4 - T1|}{T4 + T1} = \frac{|7-6|}{7+6} = \frac{1}{13} = 0.0769$$

$$D_f(\text{T4,T4}) = \frac{|T4 - T4|}{T4 + T4} = \frac{|7-7|}{7+7} = \frac{0}{14} = 0$$

Since total features in Electronic shopping is 10 and only three features absence between T1 and T4, which is standard, search and public report, $n = 3$

Enhanced Jaro Wrinkler(T1,T1) $(\text{T1,T1}) = d_{ejw}(T_1, T_1) = \frac{1}{3}\left(\frac{6}{6} + \frac{6}{6} + \frac{3-0}{3}\right) + (0(1 - d_j))$

$$= 1 + (0(1 - 1)$$

$$= 1$$

Enhanced Jaro-Winkler (T1,T4) = 0.9437 + (0.0769 * (1-0.9437) = 0.9480

Enhanced Jaro-Winkler (T4,T1) = 0.9437 + (0.0769 * (1-0.9437) = 0.9480

Enhanced Jaro-Winkler (T1,T1) = 1 + (0*(1-1)) = 1

Enhanced Jaro-Winkler (T4,T4) = 1 + (0*(1-1)) = 1

After calculation, every answer will be in similarity value between two test cases, thus the dissimilarity value is calculated by using $1- d_{ejw}$.

Prioritization Algorithm

Prioritization algorithm is often used to help determine the appropriate ranking of test cases in a prioritized test suite. In SPL, several works implemented prioritization algorithms such as Local Maximum, Global Maximum and All-yes config algorithm (Henard et al., 2014; Al-Hajjaji et al., 2016; Sanchez, Seguira & Ruiz-Cortis, 2014). These works proposed various considerations in their algorithms to accurately determine the ranking of test cases such as consideration of total maximum distance of one test case towards all test cases in the prioritized test suite and choosing two test cases with maximum distance between them as the first two test cases in the prioritized test suite. The aim of prioritization algorithms is to rearrange test cases to be able to detect high number of faults within first few test cases which is the aim of similarity-based prioritization techniques. In this paper, we provide a comparative evaluation using three sampling algorithms used in the similarity-based prioritization based on the usage of enhanced Jaro-Winkler as our similarity measure.

Local Maximum Distance

Local Maximum distance algorithm has been used by several existing works (Henard et al., 2014; Sanchez et al., 2014) in their TCP approach. Local Maximum algorithm starts with finding two unordered test cases with maximum distance between them as the first two test cases in the prioritized test suite. Next, the same process is iterated in the unordered test suite until every test case is placed in the prioritized test suite.

Global Maximum Distance

Global Maximum distance algorithm proposed by Henard et al. (2014) starts with finding two unordered test cases with maximum distance between them as the first two test cases in a prioritized test suite. Next, the algorithm will calculate the summation of total distance for each unprioritized test case inside

the prioritized test suite. This process continues until all test cases are placed in the prioritized test suite.

All-yes Config

All-yes config algorithm was proposed by Al-Hajjaji et al. (2017) to reorder the rank of each test case into the prioritized test suite. First, the algorithm will select test cases with the most features as the first test case in the prioritized test suite. The justification of choosing test cases with most features is based on the assumption that most faults will be discovered in the test cases with the most features first (Al-Hajjaji et al., 2017). Next, the algorithm will select second test case with the maximum distance towards the first test case. Lastly, the algorithm will select test cases with the maximum distance towards the test cases in the prioritized test suite with minimum distance consideration.

Eight benchmark case studies were chosen for the comparison between the three prioritization algorithms. The selection of the eight case studies are based on their usage in existing work in SPL (Sanchez, Seguira & Ruiz-Cortis, 2014; Henard et al., 2014; Al-Hajjaji et al., 2016). These case studies are widely available on Software Product Lines Online Tool (SPLOT) repository. Table 2 shows details of the benchmark case studies. APFD results from the comparison made between three prioritization algorithms described earlier is as shown in Table 3. All-yes config gained the highest average APFD scores with 80.81% followed closely by Local Maximum with 80.64% and lastly, Global Maximum with 78.67%. All-yes config prioritization algorithm gained the highest APFD scores for four out of eight FMs; meanwhile, Local Maximum scored highest APFD scores for three FMs. Other than that, Global Maximum gained highest APFD score for only one FM. These results show that All-Yes config is the best prioritization algorithm for the enhanced Jaro-Winkler similarity measure. The results also signify that All-yes config prioritization algorithm can be improved in terms of its calculation of maximum distance between test cases which will be elaborated in the following section.

Table 2

Benchmark Case Studies

Feature Model	Features	Test Cases	CTCR	Faults	Scale
Web Portal	43	19	25%	4	Medium
Video Player	71	18	0%	4	Medium
Car Selection	72	24	31%	4	Medium
Go Phone	77	14	14%	4	Medium
Model Transformation	88	28	0%	8	Medium

(continued)

Feature Model	Features	Test Cases	CTCR	Faults	Scale
Battle of Tanks	144	484	0%	12	Large
Printers	172	129	0%	16	Large
Electronic Shopping	290	24	11%	28	Large

Table 3

APFD result for comparison of three prioritization algorithms

Feature Model	Test Suite Size	No. of features	Faults Detected	APFD		
				Local Maximum	Global Maximum	All-yes Config
Web Portal	19	43	4/4	72.50	67.50	**88.10**
Video Player	18	71	4/4	91.25	82.50	**94.32**
Car Selection	24	72	4/4	78.00	**85.00**	66.00
Go Phone	14	77	4/4	85.71	96.43	**98.86**
Model Transformation	28	88	7/8	81.72	81.03	**82.00**
Battle of Tanks	484	144	11/12	**95.11**	91.61	89.03
Printers	129	172	15/16	**89.87**	88.51	84.84
Electronic Shopping	24	290	25/28	**78.69**	74.84	70.24
Average APFD				84.11	83.43	**84.17**

Enhanced All-Yes Config Prioritization Algorithm

There are two changes proposed to the existing All-yes config algorithm. The first modification is by removing some of the codes in line 10 (marked with ①) to eliminate the extra process of finding the maximum distance to the first test case. Next, this study finds two test cases with the furthest distance between them in T (set of test cases). This process is marked with ② in line 11. The enhancement done to the algorithm is as shown in Table 4. Algorithm 2 in Line 16 refers to the same algorithm by Al-Hajjaji (2016) without any modifications therefore the algorithm is not included in this section. To demonstrate the working process of enhanced All-yes config

algorithm (EA), test cases similarity distance in Table 1 is referred as an input to the prioritization algorithm. From the table, T4, T5, T6 are test cases in unprioritized test suite with the highest number of features where each test case has seven selected features. From the three test cases, T4 is selected as the first test case in the prioritized list since it is the first test case in the test suite. Next, the algorithm will iterate among test cases in the unprioritized test suite to find two test cases with the highest distance among them, in this case T3 - T6 with 0.258. The process continues to find test cases to be prioritized in the test suite with consideration of minimum distance until all test cases are placed into the prioritized test suite. The order of test case using EA algorithm is T4, T3, T6, T5, T2 and T1.

Table 4

Enhanced All-yes config prioritization algorithm

```
1: Input: T={t1, t2, t3, ....., tn} {set of test cases},
2: F (set of features}
3: Output: TCS (list of prioritized test case)
4: TCS ← [ ]
5: for i ← 1 to T.size() do
6:                for j ← i + 1 to T.size() do
7:                  AllDistance[i,j] = distance (ti , TCSj , F)
                                          ti ∈T, tj ∈T
8:                  end for
9: end for
10: Select tmax ∈ T ----------------①
11: Select ti; ti εT where max (d(ti; tj)) ---------- ②
12: TCS.add(ti)
13: T.remove(ti)
14: TestProduct(ti)
15: while T not empty do
16: ti = SelectTestCases(T, S, AllDistances) (Algorithm 2)
17:                          TCS.add(ti)
18:                          T.remove(ti)
19:                          TestProduct(ti)
20: end while
21: return TCS
```

ENHANCED SIMILARITY-BASED PRIORITIZATION EXPERIMENTATION

The main goal of this experiment is to investigate which similarity-based prioritization technique is most effective in prioritizing test cases. The technique is based on the combination of similarity measures and prioritization algorithm for test case prioritization. The experiment consists of five phases which starts with experimental setup. Then, generation of test cases will be done using ICPL sampling algorithm since it is regarded as the fastest sampling

algorithm and is able to produce acceptable size of test suite (Henard et al., 2014). Next, faults will be generated. The test cases are then prioritized using the combination of enhanced Jaro-Winkler similarity measure and enhanced All-yes prioritization algorithms. Lastly, our prioritized test suite will be evaluated based on their effectiveness criteria.

Experimental Setup

The main aim of the experiment is to evaluate the effectiveness of similarity-based prioritization technique obtained by combining enhanced string distance measure (enhanced Jaro-Winkler or EJW) and enhanced prioritization algorithm (enhanced All-yes config or EA) explained in the previous sections. In order to investigate the effectiveness, EJW is consistently used in combination with different prioritization algorithms which are EA, All-Yes Config (A), Local Maximum (LM) distance and Global Maximum (GM) distance.

This experiment is performed in controlled testing environment using a single Windows-based machine run on a laptop with 6GB RAM and Intel Core I5-3337U 1.8hz processor. Every phase of the experiment is done on this machine from the generation of the product using sampling algorithm on case studies until the testing phase of the experiment which is implemented using Java Eclipse Neon 3. Our work use the existing tool provided by Sanchez et al. (2014) which extends the SPLCAT tool for the generation of test cases. We also have added several functionalities to the tool for a complete TCP process in our research.

Generation of Test Cases using ICPL

Test cases for this experiment are generated using ICPL sampling algorithm based on input from FM as shown in Table 2. ICPL is a specialized sampling algorithm used to tackle scalability issue due to the complexity and size of industrial product lines (Johansen et al., 2012). ICPL generates covering array from the FM allowing the possibility of incorporating combinatorial interaction testing for the configuration generated from the FM. ICPL is chosen based on its improvement in terms of time in producing acceptable size of covering arrays and has low standard deviation due to its non-determinism (Johansen et al., 2012)

Fault Generation

Our work utilized the fault simulator by Ensan, Bagueri and Galsevic (2012) with 1-4 features interactions faults. The fault simulator is used by many SPL

researchers in their work to investigate the effectiveness of TCP technique (Henard et al., 2014; Al-Hajjaji et al., 2017; Sanchez, Seguira & Ruiz-Cortis, 2014). The assumption of faults spreading equally in the test cases has been used by other SPL researchers to evaluate effectiveness of TCP technique.

Prioritization of Test Suite

Test cases generated by the sampling algorithm are considered as an unprioritized test suite. Test cases inside the unprioritized test suite will be calculated based on their similarity value by using enhanced string distance measure. The result is stored inside a hashtable data structure. The stored similarity value for each test case inside hashtable data structure is iterated by the enhanced prioritization algorithm in order to rank the test cases in the prioritized test suite.

Evaluation of Effectiveness Criteria

Average Fault Detection (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 faster fault detection rate. APFD metric equation is as follows:

where $$APFD = 1 - \frac{TF_1 + TF_2 + \cdots . + TF_n}{n \times m} + \frac{1}{2_n} \quad (2)$$

T = test suite
n = test cases
= the position of the first test case exposing the faults
TF_i = number of faults exposed by test suite

The rate of faults detected refers to the amounts of faults detected in a certain level of a test suite. A good technique is the one which is capable to detect 100% of faults by using a low number of test suites. For example, technique A is capable to detect 100% of fault by using only 10% of test suite in Web Portal case study thus it is considered as a very effective technique in terms of rate of fault detection. This study provides the rate of fault detection in percentage value, the amount of fault detected is calculated every 10% where level of test suite comprised of 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100% to provide a detailed observation on result for each level.

RESULTS AND DISCUSSION

In terms of APFD scores, EA outperformed other prioritization algorithms for Web Portal, Video Player, Printer and Electronic Shopping case studies while performed second best in Go Phone case study. This proves that combination between EJW and EA is suitable to be used in similarity-based prioritization technique for most case studies. This is shown in Table 5, where the average ranked obtained by EA + EJW for benchmark case studies is the highest compared to other prioritization algorithms. Full comparison between prioritization algorithms based on their APFD results is as shown in Table 5.

Next, in terms of experiment execution time, the combination of EJW and EA gained the fastest execution time compared to three other prioritization algorithms in most of the case studies except Printers. Generally, execution time depends on the number of test cases generated for each case study. Battle of Tanks and Printers which consist of 484 and 129 test cases respectively have the highest number of test cases compared to other case studies. Theoretically, test cases from both FMs will require much longer experiment completion time. However, both case studies showed that EA is the fastest prioritization algorithm with 1568 milliseconds execution time in Battle of Tanks and second best in Printers case study with 69 milliseconds execution time. This shows that EA + EJW combination is efficient in terms of time taken for execution of large number of test cases.

Table 5

Ranks of prioritization algorithms based on APFD results

Case Study	EA	A	GM	LM
Web Portal	1	2	4	3
Video Player	1	2	4	3
Car Selection	3	4	1	2
Go Phone	2	1	2	3
Model Transformation	4	1	3	2
Battle of Tanks	3	4	2	1
Printers	1	4	3	2
Electronic Shopping	1	4	3	2
Average Rank	**2.00**	**2.75**	**2.63**	**2.25**

Table 6

Execution time based on different prioritization algorithms

Case Study	**Prioritization Execution Time (ms)**				**Scale**
	EA	**A**	**GM**	**LM**	
Web portal (19)	1	1	1	1	Medium
Video Player (18)	1	1	1	1	Medium
Car Selection (24)	1	1	1	1	Medium
Go Phone (14)	1	2	2	2	Medium
Model Transformation (28)	1	1	1	1	Medium
Battle of Tanks (484)	1568	2023	1910	1572	Large
Printers (129)	69	96	62	58	Large
Electronic Shopping (24)	3	4	3	3	Large

Lastly, for the rate of fault detected, the combination of EJW and EA obtained the best results for two case studies namely Web Portal and Video Player by using only 30% and 10% of the test suite to detect 100% of the faults. This proves the capability of EA and EJW algorithms to increase the rate of fault detected for different scale of case studies. Therefore, the proposed enhancement done on EA algorithm by considering the maximum distance of two other test cases proved to be successful and shows that omitting the first process in A algorithm as being done in EA improve the effectiveness of the algorithm.

Table 7

Rate of fault detection based on different prioritization algorithms

Case Study	EA	A	GM	LM
Web Portal	30%	40%	50%	50%
Video Player	20%	20%	30%	20%
Car Selection	60%	80%	40%	50%
Go Phone	10%	10%	10%	30%
Model Transformation	70%	60%	70%	60%
Battle of Tanks	30%	40%	30%	30%
Printers	30%	50%	40%	30%
Electronic Shopping	80%	100%	70%	80%
Average Rate	**41.25%**	**50.00%**	**42.50%**	**43.75%**

CONCLUSION

The enhanced string distance technique, EJW takes consideration of the features inside SPL for the modification of its similarity distance calculation. Modification was done on original Jaro distance and Winkler equation by incorporating Degree of Difference, D_f The implementation of D_f is important to provide a much more accurate and consistent similarity value since the original Jaro-Winkler equation uses prefix in their equation which is not suitable for configuration of test cases such as in SPL.

Additionally, for the enhanced prioritization algorithm, EA improves the existing algorithm by eliminating the first process in A of selected test case which has the highest features. Instead, a new process is added by selecting two test cases with maximum distance between them in the unprioritized test suite. The proposed enhancement also supports the aim of similarity-based prioritization technique by placing test cases with higher distance between each other first in the test suite.

Lastly, better results were obtained from the combination of EJW and EA in terms of APFD scores, execution time and rate of fault detection. Thus, the combination between EJW string distance and EA prioritization algorithm produced a complete synergized similarity-based prioritization technique which improves the effectiveness of SPL testing process. It is hoped that the proposed technique could facilitate and ease SPL testers process in completing their daily work.

ACKNOWLEDGMENT

This work is supported by the Research University Grant (GUP), Universiti Teknologi Malaysia (UTM) with Cost Centre ID Q.J130000.2528.15H44, 2017.

REFERENCES

Al-Hajjaji, M., Lity, S., Lachmann, R., Thüm, T., Schaefer, I., & Saake, G. (2017). Delta-oriented product prioritization for similarity-based product-line testing. *Proceedings of the 2nd International Workshop on Variability and Complexity in Software Design* (pp. 34-40). IEEE Press.

Al-Hajjaji, M., Thüm, T., Lochau, M., Meinicke, J., & Saake, G. (2016). Effective product-line testing using similarity-based product prioritization. *Journal of Software & Systems Modeling* (SoSym), 1-23.

Choi,K., Szakal, B., Chen, Y. H., Branzei, D., & Zhao, X. 2010 The Smc5/6 complex and Esc2 influence murtiple replication-associated recombination processes in Saccharomyces cerevisiae. *Mol Biol Cell*, *21*(13),2306-2314. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2893993/

Chvatal, V. (1979). A greedy heuristic for the set-covering problem. *Journal of Mathematics of Operations Research*, *4*(3), 233-235.

Cohen, W., Ravikumar, P., & Fienberg, S. (2003). A comparison of string metrics for matching names and records. *Proceedings of the Ninth International Conference on Knowledge Discovery and Data Mining (KDD) at the International Workshop on Data Cleaning and Object Consolidation* (pp. 73-78).

Cohen, M., Dwyer, M., & Shi, J. (2008). Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Transactions on Software Engineering*, *34*(5), 633-650.

Devroey, X., Perrouin, G., Cordy, M., Schobbens, P. Y., Legay, A., & Heymans, P. (2014). Towards statistical prioritization for software product lines testing. *Proceedings of the Eighth International Workshop on Variability Modeling of Software-Intensive Systems* (VaMos) (pp. 1-7). ACM.

Devroey, X., Perrouin, G., Cordy, M., Samih, H., Legay, A., Schobbens, P. Y., & Heymans, P. (2017). Statistical prioritization for software product line testing: An experience report. *Journal of Software & Systems Modeling* (SoSym), *16*(1), 153-171.

Devroey, X., Perrouin, G., Legay, A., Schobbens, P. Y., & Heymans, P. (2016, January). Search-based similarity-driven behavioural SPL testing. *Proceedings of the Tenth International Workshop on Variability Modeling of Software-intensive Systems* (pp. 89-96). Salvadore, Brazil.

do Carmo Machado, I., Mcgregor, J. D., Cavalcanti, Y. C., & De Almeida, E. S. (2014). On strategies for testing software product lines: A systematic literature review. *Journal of Information and Software Technology*, *56*(10), 1183-1199.

Ensan, F., Bagheri, E., & Gašević, D. (2012). Evolutionary search-based test generation for software product line feature models. *Proceedings of the Advanced Information Systems Engineering* (pp. 613-628).

Garvin, B., Cohen, M., & Dwyer, M. (2011). Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Journal of Empirical Software Engineering*, *16*(1), 61–102.

Henard, C., Papadakis, M., Perrouin, G., Klein, J., Heymans, P., & Le Traon, Y. (2014). Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Transactions on Software Engineering*, *40*(7), 650-670.

Johansen, M. F., Haugen, Ø., Fleurey, F., Eldegard, A. G., & Syversen, T. (2012). Generating better partial covering arrays by modeling weights on sub-product lines. *Proceedings of the International Conference on Model Driven Engineering Languages and Systems* (pp. 269-284).

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. (1990). Feature-oriented domain analysis (FODA) feasibility study. *Technical Report CMU/SEI-90-TR-21*. Software Engineering Institute, Carnegie Mellon University.

Oster, S., Zorcic, I., Markert, F., Lochau, M., (2011). MoSo-PoLiTe–Tool support for pairwise and model-based software product line testing. *Proceedings of the Workshop Variability Modelling of Software-intensive Systems* (VaMoS) (pp. 79–82). ACM.

Pohl, K., & Metzger, A. (2006). Software product line testing. *Communications of the ACM*, *49*(12), 78-81.

Sahak, M., Jawawi, D. N., & Halim, S. (2017). An experiment of different similarity measures on test case prioritization for software product lines. *Journal of Telecommunication, Electronic and Computer Engineering* (JTEC), *9*(3-4), 177-185.

Sánchez, A. B., Segura, S., & Ruiz-Cortés, A. (2014, March). A comparison of test case prioritization criteria for software product lines. *Proceedings of the IEEE Seventh International Conference on Software Testing, Verification and Validation*, (pp. 41-50). IEEE. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download. doi:10.1.1.715.4870

Tumeng, R. (2017). *Test case prioritization with requirements change using string metrics* (Unpublished master's thesis). Universiti Teknologi Malaysia, Skudai.

Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. *Journal of Software Testing, Verification and Reliability*, *22*(2), 67-120. doi: 10.1002/000

Zamli, K. Z., Klaib, M. F. J. Younis, M. I. & Yeh, O. H. (2010). G2Way: A pairwise test data generation strategy with automated execution support. *Journal of Information and Communication Technology*, *9*, 59-85.