Journal of Theoretical and Applied Information Technology
30th June 2017. Vol.95. No 12
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A HYBRID WEIGHT-BASED AND STRING DISTANCES USING PARTICLE SWARM OPTIMIZATION FOR PRIORITIZING TEST CASES

[1] MUHAMMAD KHATIBSYARBINI, [2] MOHD ADHAM ISA, [3] DAYANG NORHAYATI ABANG JAWAWI

Department Software Engineering, Faculty of Computing, Universiti Teknologi Malaysia

81310 Skudai, Johor, Malaysia

E-mail: [1] fkmuhammad4@gmail.com, [2]mohdadham@utm.my

ABSTRACT

Regression testing is concerned with testing the modified version of software. However, to re-test entire test cases require significant cost and time. To reduce the cost and time, higher average percentage fault detection (APFD) rate and faster execution to kill fault mutant are required. Therefore, to achieve these two requirements, an improvement to existing Test Case Prioritization (TCP) technique for a more effective regression testing is offered. A weight-hybrid string distance technique and prioritization using particle swarm optimization (PSO) is proposed. Distance between test cases and weight for each test case, and hybridization of both values for weight-hybrid string distance are calculated. This experiment was evaluated using Siemens dataset. Result obtained from this experiment shows that weight-hybrid string distance is capable of improving APFD values whereby APFD value for hybrid TFIDF-JC is equal to 97.37%, which shows the highest improvement by 4.74% as compared to non-hybrid JC. Meanwhile, for percentage of test cases needed to kill 100% fault mutants, hybrid TFIDF-M yields the lowest value, 22.88%, which shows a 76% improvement as compared to its non-hybrid string distance.

Keywords: *Software testing, Regression testing, Test case prioritization, Particle Swarm Optimization, String Distance*

1. INTRODUCTION

In software development process, software maintenance activity consumes a longer execution time and can be the most expensive phase [1]. One of the most crucial stages in maintenance activity is testing phase, known as regression testing, which is executed with a specific end goal to ensure the adjustment or modification in the system does not influence existing functionality. In other words, regression testing is a testing activity which would only be performed if there are changes acted upon a system. It determines whether the new system operates as expected when compared to functioning old system's version. In the work of Yoo and Harman [2], various diverse approaches were examined to augment the importance of the accumulated test suite in regression testing. Those studies were classified into three domains; Minimization, Selection and Prioritization. Test suite minimization (TSM) approach intends to distinguish repetitive experiments and to eliminate them from the test suite execution with a specific end goal to decrease the quantity of tests to run [3]. Minimization or name as other name which is 'test suite reduction', implying that the disposal is perpetual. Test case selection (TCS) approach also aims to decrease the quantity of test cases to be executed, however the mainstream of selection approach is based on modification-aware method [4]. TCS tries to recognize the test cases which would be important to the latest changes acted upon a system.

Lastly, test case prioritization (TCP) main goals are to order the whole test suite to attain early optimization based on preferred properties [5], [6]. It gives a technique to execute test cases of highest significance first according to some measure, and produce some aids, such as providing earlier fault disclosure and criticism to the testers. In TCP, test cases are re-ordered optimally as compared to the un-ordered generated test suite according to a particular purpose in a manner whereby the test cases that serve the purpose will be given the highest priority [7].

We took the definition of Test Case Prioritization problem which was proposed by Elbaum *et al.* [8] into consideration which is stated as follows.

Given: *T*, is a test suite; *PT*, is the set of permutations of *T*; *f*, a function from *PT* to the real number. Find *T* ∈ *PT* such that

$$(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')] \quad ..(1)$$

In this definition, PT serves as the set of all possible sequences of *T*, while *f* is the function when implemented to any of the sequences, yields an *award value* for that particular sequence. In short, the explanation expects that greater values of the results are more preferable than the inferior or the smaller values ones. There are various conceivable objectives when alluding to prioritization in this context. Elbaum *et al.* [8] also stated some of the goals in their study which are:

1) To upsurge or improve the average percentage fault detection (APFD) values when executing entire test suite.
2) To kill all fault mutants at a faster pace when executing a test suite.

Over time, researchers have proposed numerous approaches for TCP. In code-based TCP, test cases are prioritized by utilizing source code information of the software system. A study carried out by Catal and Mishra [9] revealed that the most investigated prioritization method is coverage-based, which is related to code-based prioritization. The downside of code-based prioritization is that code knowledge is needed in order to prioritize test cases, which means prioritization cannot begin until the source code is available [10]. Another drawback of code-based approaches is that most of them are language dependent [10], so testing process will become more complicated in cases where the program is written in various programming languages. There are quite a number studies in TCP that focused on regression problems with solution relying on both old and new system codes but very few have tried to utilize test cases generated associated with system changes.

In this paper, a new weighted hybrid string-based TCP technique is proposed. This technique utilizes string inputs of test cases without the consideration of the program source code. String-based prioritization calculates the string distance between test cases and then prioritizes based on the calculated distance. String distance were measured based on the string or terms arrangements and also their character sequences [11]. There quite a number of string distance types and four are used in this experiment; Manhattan, Levenshtein, Cosine Similarity, and Jaccard Coefficient.

However, using only string distance values, the possibility to have similar distance is quite high and may affect prioritization process. Therefore, to overcome the problem, hybridization of all four string distances with a weighting scale for text document was carried out. As a result, the distances calculated for each test case to the other will be refined with their incorporating weight. This will make a major difference in test case execution sequence which can increase the APFD rate and kill all fault mutants at a faster pace. The proposed technique is validated by performing a comparative study experiment using Siemens programs. The evaluation is based on APFD values whereby the greater the values, the superior the technique. Meanwhile, for fault mutant killed performance; smaller values are better which are measured based on percentage of test cases required to achieve 100% mutants killed.

The rest of the paper is structured as follows: Section 2 elaborates related work in TCP based on string distance. Section 3 provides the preliminaries which is an overview of related string distance, weighting document scale, and prioritization algorithm for the proposed work. In Section 4, a controlled experiment for the proposed work is presented and illustrated. Section 5 reports on our result and discussion, and Section 6 summarize the conclusion of this experiment.

2. RELATED WORK

As prioritization on test cases had only gone through test case selection in early studies [5], TCP was then suggested and assessed in a further broad context. In real world situation, it is quite hard to determine which tests will detect faults. Hence, it justifies the notion behind test case prioritization approaches to have other backups, expecting that a certain number of backup approaches will end in boosting fault discovery in different ways. There were several TCP approaches that had been anticipated and applied in the previous work. As many as eight broad approaches were described by Singh [12]. The TCP techniques approaches were based on their commonalities in selection procedure, input type, and output type.

Work by Ledru [13] presented test case prioritization based on string distances in detecting the strongest mutants with Manhattan distances as the best choice in their findings. In their proposed work, they compared four distances namely Cartesian, Hamming, Levenshtein, and Manhattan distances. These distances are measured in terms of characters distances when characters are replaced by new characters. The distances are quantified in terms of characters difference with prioritization using Greedy Algorithm.

Meanwhile, the work by Bo Jiang [14] presented Levenshtein or Edit distance in detecting the strongest mutants in Linux benchmark program. In their work, they focused on prioritization algorithm and claimed Levenshtein as the most suitable string metric.

From both reported works, prioritized test cases using string metrics have promising average percentage of fault detection (APFD) values compared to random ordered ones. The average of APFD rank for all three string metrics distance were the same except for Cartesian which had the worst performance [13].

3. PRELIMINARIES

3.1 The String Distance for TCP

String distance were measured based on the string or terms arrangements and also their character sequences [11]. A string distance is a metric that can quantify likeness or divergence between two contents of strings for surmised string coordinating or examination. There are two type of string distances; character-based and term-based [11]. Both types have various string metrics or distance calculations.

$$\sum_{i=1}^{n} |x_i - y_i|$$

Manhattan Distance ...(2) [13]

Character-based string metric utilizes each character in a string and compares it with the character in other string. For example, Manhattan distance (Equation 2) where string of size *s* can be seen as a path of characters in an *s*-dimensional space, and the characters can be correlated to their ASCII code. Manhattan distance is equal to the absolute difference of the ASCII value of strings. The idea of comparing apply with the concept where char '0' were used to fill the shorter string when the strings does not have equal length [13].

$$s_{Cos} = \frac{\sum_{i=1}^{d} P_i Q_i}{\sqrt{\sum_{i=1}^{d} P_i^2}\sqrt{\sum_{i=1}^{d} Q_i^2}}$$

Cosine Similarity ...(3) [15]

Meanwhile for term-based, the calculation of the string metric defined is based on the whole term of the string compared to the term in other string. For example, Cosine Similarity (Equation 3) represents documents as a vector, the likeness of two documents bear a resemblance to their vectors distances association. These resemblances were then measured and call as cosine similarity with the calculation involved were the cosine value of the angle or distance between vectors. Cosine similarity is a standout amongst the most famous similarity measures connected to text content reports, for example, in various data recovery applications and grouping [16]. For each vectors characterizes a string with its frequency within the text, which cannot be a negative number. As the consequence, the cosine similarity measured in a positive value and restricted in the middle of zero and one value.

3.2 The Term Frequency Inverse Document Frequency (TFIDF)

"Term frequency–inverse document frequency" (TFIDF) is a standout amongst the most ordinarily utilized term weighing schemes in data retrieval methods. As for its well-known technique, TFIDF has been regularly applied in an experimental evaluation[17]. TFIDF is also recognized as a statistical measurement with the intention to reveal how significant a term is within a file in its collection or in other words, a weighing scheme for text documentation.

For the formulation of term frequency tf (Equation 4), the calculation start with the use of the selected term or string frequency in a document, in simplest word, the amount of the term t take place within a document d. While, the formula for inverse document frequency idf (Equation 5), deal with the significance of the term in the whole documents pool.

$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} tf_{ik}} . \operatorname{Log}(f_{ik})$$

... (4)

$$idf_k = \log \frac{N}{n_k}$$

... (5)

This inverse document frequency measures the amount of information carried by terms. TFIDF is formed by the multiplication of tf.idf with some heuristics modifications.

3.3 The Algorithm for TCP

The implementation of artificial intelligence or search-based approach in TCP is not limited to specific strategies only. Within TCP itself, there are several algorithms used including Genetic Algorithm (GA) [18]–[26], Greedy [27], [28], Ant-Colony [29]–[31], Particle Swarm Optimization [32], [33], and others [14], [34]. Several observations are noted for artificial intelligence (AI) utilization. First, there are many publications on AI application which is used to solve different problems contexts. Second, empirical data is easily available for AI experimental setup. This encourages researcher to execute and compile results using search-based approach.

Search-based prioritization approach has quite a number of implementation algorithms such as Genetic Algorithm (GA) [18]–[25], Greedy [27], [28], Ant-Colony [29]–[31], Particle Swarm Optimization [32], [33], and others [14], [34]. Experiment by Li [28] revealed that GA application approach works poorer when compared to a greedy algorithm on computer-generated data. However, the application of a search-based algorithm may differ, based on the selected test suite, input criteria, fitness function, and others. While the collected result showed a major benefit of GA application in TCP approaches, there are certain disadvantages that exist, such as, execution time is a vast anxiety for GA applications and they are typically slow in the process of completion [21].

In this proposed work, rather than using the program code, the test cases generated are utilized for prioritization based on their inputs to calculate string distance/similarity between test cases. Related work by Ledru [13] using Greedy and Jiang [35] using Local Beam Search showed promising results in term of average percentage fault detected in prioritizing the input based on the test cases. Therefore, since this experiment will utilize the inputs of the test cases, PSO is intended to be used as our prioritization algorithm since PSO has the best efficiency [36] in getting the shortest string metric distance between test cases.

4. EMPIRICAL STUDIES

In this section, a weight-hybrid string distance technique and prioritization using particle swarm optimization is proposed to improve regression testing by achieving higher average percentage fault detection (APFD) rate and faster fault mutant killed.

4.1 The Experiment Setup

In this section, the experiment setup for test case prioritization using Siemens dataset [37] is described. The Siemens Suite is a well-used benchmark programs software appeared in numerous regression testing literature. Siemens suite has several small software programs using C as its main language of programming. The software consists of small programming code with code lines, between 173 and 565 (141 to 512 without code make-up). Each program comes with a reference version, and several versions with seeded faults. If a test case is able to reveal different output on the reference version and the version with seeded fault, this considers that a mutant is killed.

For this experiment three datasets of a similar application but different version of language are utilized. Table 1 below shows a clear overview of the datasets used.

Table 1: Overview of Datasets

Dataset Name	*Programming Language*	*Fault Matrix*
tcas	C	Have Different Fault Matrix
jtcas	Java	
cstcas	C#	

From Table 1, the dataset used is *tcas* which is originally a C program of an aircraft collision avoidance system. It takes 12 integer inputs and produces one output. The program came with one base version and 41 faulty versions with 1608 test cases. The fault matrix is produced by executing all test cases on all 41 faulty versions and compared against their base version. Even as all the datasets are for the same application, the fault matrix for each version of the application is different.

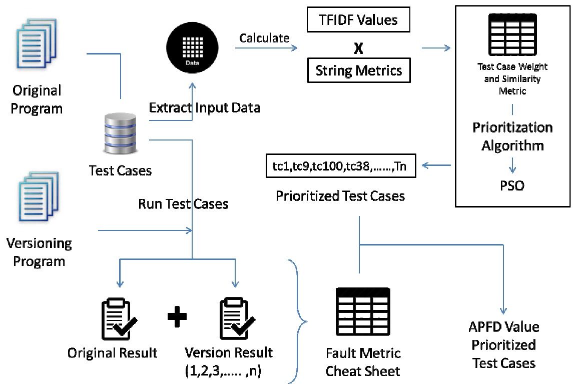

Figure 1: Experiment Prioritization Design

Figure 1 illustrates the experiment flow for this paper. The experiment can be divided into three phases; Information Extraction phase, String Distance and Prioritization phase, and Evaluation phase. For information extraction phase, the phase starts with extracting all the test cases and their inputs. The extracted inputs for each test case were put into different text document to ease the calculation needed in the next phase. Then, the original and versioning programs were run through the extracted test cases to get their respective result. The results for each version were compared with original result to produce s fault matrix sheet to be used in evaluation phase.

The next phase is the calculation of string distance and prioritization phase. The string distances between test cases are calculated based on the extracted inputs and populated into test case distance matrix. Within this calculation part, each test case is weighted using TFIDF method. The proposed hybridized string distance is also carried out here. Upon completion of the calculation, the test case distance matrix is then prioritized using particle swarm optimization (PSO) to produce prioritized test suites with the shortest distance possible in total.

Finally, in the evaluation phase, the prioritized test suites are evaluated by calculating their APFD rate based on fault matrix of their program. The fault matrix was obtained in earlier stage by executing all test cases for all versioning programs and comparing the outputs against the original program. The results is compared to produce the fault matrix sheet. The percentage of test cases required to kill 100% fault mutants is also calculated.

4.2 The Proposed Hybrid String Distance

This experiment will utilize the inputs of the test cases. The inputs will be calculated for their similarity between test cases using string metric. The string distance to be used in this experiment consists of two character-based string distances; Manhattan distance (Equation 2) evaluated by Ledru [13] and Levenshtein distance (Equation 6) used by Bo Jiang [14], and two term-based string distances; Cosine Similarity distance (Equation 3) and Jaccard Coefficient distance (Equation 7).

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if} \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein Distance … (6) [14], [38]

$$s_{Jac} = \frac{\sum_{i=1}^{d} P_i Q_i}{\sum_{i=1}^{d} P_i^2 + \sum_{i=1}^{d} Q_i^2 - \sum_{i=1}^{d} P_i Q_i}$$

Jaccard Coefficient …(7) [15]

Since Manhattan and Cosine Similarity already been explained in preliminaries section, for Levenshtein distance, the value is calculated by counting the minimum number of operations required to transform one string into another. Jaccard in the other hand is calculated based on the number of mutual terms in compared to the amount of all exclusive terms in both strings [11]. A hybrid string distance with TF-IDF is then proposed. TFIDF is a weighing scheme for text documentation. Table 2 shows the brief idea of how the string distance values are hybridized.

Table 2: The Overview Idea Hybrid String Distance

TFIDF Values for Each Test Case	*String Metric Value Between Test Cases*	*Hybrid String Metric with TFIDF*
Weight of TC1 = W1	Distance/ Similarity between TC1 – TC2 = D1,2	D1,2 . W2
.	.	.
.	.	.
.	.	.
Weight of TCn = Wn	Distance/ Similarity between TCn – TCn+1 = Dn,n+1	Dn,n+1 . Wn+1

As shown in Table 2, the formulation of the hybrid string distance is obtained by multiplying the value of the string distance with the weight of the next test cases. The idea to use string distance is inspired by the work of Bo Jiang [14] where it has been reported that the implementation of different distances may produce distinct results. As the prioritization is based on the difference between two points of test cases, instead of using only one metric, this study proposes to add the weight of next test case point as the second metric to be considered. This inspires the authors to hybrid every string distance highlighted earlier with TFIDF which is the weighting scale for text document to yield a better priority value for each test case.

Based on the overview of the string distance and TFIDF weight hybridization idea, the authors produced four new hybridized TFIDF string distances with the formulation as shown in Table 3.

Table 3: The Formulation of String Distance

String Distance	*Equation*	
	Non-Hybrid	*Hybrid TFIDF*
Manhattan	(2)	(2) × (4) × (5)
Levenshtein	(6)	(6) × (4) × (5)
Cosine Similarity	(3)	(3) × (4) × (5)
Jaccard Coefficient	(7)	(7) × (4) × (5)

4.3 Particle Swarm Optimization Algorithm

Particle swarm optimization algorithm used in this experiment is shown in Figure 2.

```
Begin
Initial   No of Particle, n, Velocity, v, Target, t, Best Dist., b,
          Max Iterate, m, Test Cases, s, Path, p, Total Dist., d,
Start     n Particle randomly start in different point
IF        Iteration less than m and target distance, t not reached
          n Particle move randomly to next point with velocity v
          n Particle move until last point (s)
          Calculate d for each n particle p and get the shortest
          IF        b > Shortest d among n particle p
                    b = the shortest
IF        iterate equal to m or b < t
          Return p of b
ELSE      repeat line 4
END
```

Figure 2: PSO Algorithm for String Distance Prioritization

The prioritization starts with by initializing the number of swarm particles, velocity of swarm, maximum iteration, target distance, best distance, size of test cases, and the path. The number of swarm particles and the velocity are used to adjust the execution time of the prioritization process. Target distance is used as a target of shortest path to search within the test suites. Iteration limits are set to avoid an endless iteration of prioritization. Figure 3 shows some fixed values for the prioritization in *tcas* dataset:

Number of Swarm Particle	= 50
Velocity	= 10
Size of Test Cases (Population)	= 1608
Maximum Iteration	= 5000

Figure 3: PSO Algorithm Parameter Setting

As shown in Figure 3, the number of swarm particles and velocity are set at high values as the size of the population is huge. The maximum iteration is set at 5000 to avoid any missed paths. It takes around 20 minutes for prioritization iteration to reach maximum. The target distance setting varies based on the string distance/similarity metrics calculated.

4.4 Metric Used for Comparison

It is essential for any approaches proposed in test case prioritization to perform metric measurement to assess their effectiveness. This process is important to measure the efficacy of the proposed approach in prioritizing test cases and to benchmark its effectiveness against other existing approaches. One of the metrics used in evaluation of prioritization effectiveness is Average Percentage of Faults Detected (APFD). APFD is a metric used to quantify how rapid a arranged and optimized test suite can discovers defects [5], [8]. The result of APFD values ranged in between zero to 100 where greater value indicate better fault revealing rate. The equation for calculating APFD value is shown as follows.

$$APFD = 1 - \frac{TF_1 + TF_2 + \ldots + TF_n}{n \times m} + \frac{1}{2n} \quad \ldots (8)$$

Where T is a test suite containing n test cases, F is a set of m faults revealed by T. TF_1 is the first test case in T' ordering of T which reveals fault i and the APFD value of T' is calculated using the Equation 8.

5. RESULTS AND DISCUSSION

The experiment starts with the calculation of string metrics in this experiment where two are character-based string metrics; Manhattan (M) and Levenshtein (L), while another two are term-based string metrics; Cosine Similarity (CS) and Jaccard Coefficient (JC). The authors then hybridize those string metrics with term frequency–inverse document frequency (TFIDF), which is a numerical statistic that is intended to reflect how important a word is to a document in a collection or in other words, a weighing scheme for text documentation. The values of APFD rate for each dataset are recorded in Table 4. For mutant killed performance, the percentage of test cases needed to kill all faults mutant are recorded in Table 5.

Table 4: Average Percentage Fault Detected (APFD) Rate

String Distance	*String Based*	*APFD Rate (%)*		
		tcas	*jtcas*	*cstcas*
M	Character-Based	92.39	92.70	92.80
L		90.13	91.50	90.75
CS	Term-Based	91.80	92.35	92.22
JC		92.22	92.99	92.96
TFIDF-M	Weight-hybrid Character-Based	95.87	95.61	94.48
TFIDF-L		91.73	92.71	92.40
TFIDF-CS	Weight-hybrid Term-Based	92.79	94.48	93.24
TFIDF-JC		96.19	95.34	97.37

Table 4 shows APFD rate values for each string metric, where the higher is the value, the better is the string metric. From Table 4, there are two string metrics for each string-based. All the string metrics can measure distance or similarity by just minimally alternating their formulation of measurement. Among the non-hybrid string distances, JC a term-based string distance has the highest APFD value which is 92.96% for *cstcas* dataset. Hybridized TFIDF-JC scored the highest APFD value which is 97.37% for *cstcas* dataset as well. The hybrid TFIDF-JC yields the highest value attributed to the input type for the test cases. The input for this dataset application takes 12 integer inputs and produces one output test case. A small change of character within the integers may have significant differences between the integers. Therefore, prioritizing the whole term where it considers every whole integer should be better rather than prioritizing based on each character on an integer itself. The idea of weighing TFIDF is to calculate the weight of each test case based on the frequency of term used, giving an extra priority to the input values within the test case based on their occurrences has resulted in better APFD score as compared to all to non-hybrid string distances.

Table 5: Mutant Killed Performance

String Distance	*Percentage Test Cases Used to Kill 100% Fault Mutants*		
	tcas	*jtcas*	*cstcas*
M	92.22	92.22	92.22
L	76.18	69.97	76.18
CS	65.80	65.80	65.80
JC	42.60	42.60	42.60
TFIDF-M	22.88	22.88	22.88
TFIDF-L	47.08	37.19	37.19
TFIDF-CS	67.60	49.25	49.25
TFIDF-JC	47.33	47.33	34.89

From Table 5, the data indicates the percentage of test cases needed for each string metric to kill 100% fault mutants or in other words to detect all faults. Among the non-hybrid string distances, JC outperforms other string distances by having the lowest percentage of test cases needed to kill all mutants which is 42.60% for *cstcas* dataset. Both character-based string distances obtain higher values against term-based. Numerous difference in characters within a term may have resulted in a higher difference value in the character-based string metric which may have affected the sequence of the prioritized test cases. The advantage of term-based over character-based sting distances is that they will produce over calculate distance since they consider the whole term as one value to be calculated.

However, this is not the case for hybrid weight, where the percentage values for character-based hybridized string distances yield a significantly reduced percentage of test cases needed to kill 100% fault mutants compared to term-based hybridized string distances. The hybrid TFIDF-M has the lowest value with 22.88% among all string distances which shows a huge difference with 75.19% improvement as compared to its non-hybrid string distance which scored 92.22%. This result is attributed to TFIDF which only focuses on the term frequency in a document within a collection or corpus. It does make a good complement for character-based string distances which lack whole terms values.

From the overall results of this experiment, the authors conclude that hybrid string distance does have an improvement in APFD rate values and mutant killed performance. As for the string distance, Jaccard Coefficient seems to be an ideal string distance compared to the other three string distances, as its result for APFD rate and percentage of test cases needed to kill all fault mutants showed promising scores both for; before hybridization and after hybridization with TFIDF.

6. CONCLUSION

Within this paper, we proposed a hybrid string metric test case prioritization using particle swarm optimization. This approach utilized available test cases information such as inputs for test cases. By this, the prioritization can be executed earlier for initial testing since the program/software codes are not required.

The authors also elaborated particle swarm optimization algorithm and performed the experiment using four non-hybrid string metrics and four hybrid string metrics. The experiment was performed on Siemens dataset. The empirical study result show that the test suites prioritized by the proposed hybrid string technique yields better result in terms of Average Percentage Fault Detection (APFD) and mutant killed performance. For the highest APFD rate, TFIDF-JC shows 4.74% improvement from 92.96% to 97.37% over its non-hybrid string metric, while for mutant killed performance TFIDF-M shows 75.19% improvement from 92.22% to 22.88% over its non-hybrid string metric. The hybrid string metric yields better scores since, the idea of weighing scheme, TFIDF, provides an extra priority for each test case during prioritization, which makes it an ideal complement for character-based string distances which lack whole terms values.

The acquired experimental result also showed that Jaccard Coefficient, a term-based string metric, hybridized with TFIDF gives better APFD values compared to the others. However, Manhattan, a character-based string metric, hybridized with TFIDF outperformed the others in terms of percentage of test cases needed to kill 100% fault mutants. For upcoming work, it seems to be an interesting work to hybrid term and character-based string metrics and combine them with a weighing scheme for text document.

ACKNOWLEDGEMENT

The work is fully funded by Fundamental Research Grant Scheme, vote number 4F836 and Research University Grant Scheme, vote number 11H86 under the Ministry of Education. We would also like to thank the members of Embedded & Real-Time Software Engineering Laboratory (EReTSEL), Faculty of Computing, UTM for their feedback and continuous support.

REFRENCES:

[1] G. J. Myers, T. M. Thomas, and C. Sandler, *The Art of Software Testing*, vol. 1. John Wiley & Sons, 2004.

[2] S. Yoo and M. Harman, “Regression Testing Minimisation, Selection and Prioritisation : A Survey,” *Test Verif Reliab*, vol. 0, pp. 1–7, 2007.

[3] D. Jeffrey and N. Gupta, “Improving fault detection capability by selectively retaining test cases during test suite reduction,” *IEEE Transactions on Software Engineering*, vol. 33, no. 2, pp. 108–123, Feb. 2007.

[4] S. Elbaum, P. Kallakuri, A. G. Malishevsky, G. Rothermel, and S. Kanduri, “Understanding the effects of changes on the cost-effectiveness of regression testing techniques,” *Journal of Software Testing, Verification and Reliability*, vol. 12, no. 2, pp. 65–83, 2003.

[5] G. Rothermel, R. H. Untch, C. C. Chu, and M. J. Harrold, “Test case prioritization: an empirical study,” in *Software Maintenance, 1999. (ICSM ’99) Proceedings. IEEE International Conference on*, 1999, pp. 179–188.

[6] S. Elbaum, A. G. Malishevsky, and G. Rothermel, “Test case prioritization: a family of empirical studies,” *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[7] Y. Singh, “Systematic Literature Review on Regression Test Prioritization Techniques Difference between Literature Review and Systematic Literature,” *Informatica*, vol. 36, pp. 379–408, 2012.

[8] S. Elbaum, A. Malishevsky, and G. Rothermel, *Prioritizing test cases for regression testing*. 2000.

[9] C. Catal and D. Mishra, “Test case prioritization: a systematic mapping study,” *Software Quality Journal*, vol. 21, no. 3, pp. 445–478, 2012.

[10] A. Mahdian and A. Andrews, “Regression testing with UML software designs: a survey,” *Journal of Software*, 2009.

[11] W. Gomaa and A. Fahmy, “A survey of text similarity approaches,” *International Journal of Computer*, 2013.

[12] A. Kumar and K. Singh, “A Literature Survey on test case prioritization,” *Compusoft*, 2014.

[13] 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.

[14] B. Jiang and W. K. Chan, “Input-based Adaptive Randomized Test Case Prioritization,” *J Syst Softw*, vol. 105, no. C, pp. 91–106, 2015.

[15] S. Cha, “Comprehensive survey on distance/similarity measures between probability density functions,” *City*, 2007.

[16] A. Huang, “Similarity measures for text document clustering,” *Proceedings of the sixth new zealand computer*, 2008.

[17] A. Aizawa, “An information-theoretic perspective of tf–idf measures,”

Information Processing & Management, vol. 39, no. 1, pp. 45–65, Jan. 2003.

[18] R. Maheswari and D. Mala, "Combined Genetic and Simulated Annealing Approach for Test Case Prioritization," *Indian Journal of Science and Technology*, 2015.

[19] Y. Lou, D. Hao, and L. Zhang, "Mutation-based test-case prioritization in software evolution," *2015 IEEE 26th International Symposium on Software Reliability Engineering, ISSRE 2015*, pp. 46–57, 2016.

[20] F. Yuan, Y. Bian, Z. Li, and R. Zhao, "Epistatic Genetic Algorithm for Test Case Prioritization," *International Symposium on Search Based*, 2015.

[21] C. Catal, "On the application of genetic algorithms for test case prioritization: a systematic literature review," *Proceedings of the 2nd international workshop on*, 2012.

[22] A. Kaur and S. Goyal, "A genetic algorithm for fault based regression test case prioritization," *International Journal of Computer Applications*, vol. 32, no. 8, pp. 975–8887, 2011.

[23] W. Jun, Z. Yan, and J. Chen, "Test case prioritization technique based on genetic algorithm," *Internet Computing & Information*, 2011.

[24] S. Sabharwal, R. Sibal, and C. Sharma, "Prioritization of test case scenarios derived from activity diagram using genetic algorithm," *2010 International Conference on Computer and Communication Technology, ICCCT-2010*, pp. 481–485, 2010.

[25] K. Deb, S. Pratab, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NGSA-II," *IEEE Transactions on Evolutionary Computing*, vol. 6, no. 2, pp. 182–197, 2002.

[26] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," *IEEE Transactions on Software Engineering*, vol. 32, no. 9, pp. 733–752, 2006.

[27] Z. Li, M. Harman, and R. M. Hierons, "Search Algorithms for Regression Test Case Prioritization," *IEEE Transactions on Software Engineering*, vol. 33, no. 4, pp. 225–237, 2007.

[28] S. Li, N. Bian, Z. Chen, and D. You, "A simulation study on some search algorithms for regression test case prioritization," *2010 10th International*, 2010.

[29] K. Solanki, Y. Singh, S. Dalal, and P. Srivastava, "Test Case Prioritization: An Approach Based on Modified Ant Colony Optimization," *Emerging Research in*, 2016.

[30] D. Gao, X. Guo, and L. Zhao, "Test case prioritization for regression testing based on ant colony optimization," *Software Engineering and Service*, 2015.

[31] T. Noguchi, H. Washizaki, and Y. Fukazawa, "History-Based Test Case Prioritization for Black Box Testing Using Ant Colony Optimization," *2015 IEEE 8th*, 2015.

[32] A. K. Joseph, G. Radhamani, and V. Kallimani, "Improving test efficiency through multiple criteria coverage based test case prioritization using Modified heuristic algorithm," in *2016 3rd International Conference on Computer and Information Sciences (ICCOINS)*, 2016, pp. 430–435.

[33] M. Tyagi and S. Malhotra, "Test case prioritization using multi objective particle swarm optimizer," in *2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014)*, 2014, pp. 390–395.

[34] S. Eghbali and L. Tahvildari, "Test Case Prioritization Using Lexicographical Ordering," *IEEE Transactions on Software Engineering*, vol. 5589, no. January, pp. 1–1, 2016.

[35] B. Jiang and W. Chan, "Input-based adaptive randomized test case prioritization: A local beam search approach," *Journal of Systems and Software*, 2015.

[36] A. W. Mohemmed, N. C. Sahoo, and T. K. Geok, "Solving shortest path problem using particle swarm optimization," *Applied Soft Computing*, vol. 8, no. 4, pp. 1643–1653, Sep. 2008.

[37] "Software-artifact Infrastructure Repository: Home." [Online]. Available: http://sir.unl.edu/portal/index.php. [Accessed: 20-Mar-2017].

[38] D. Gusfield, *Algorithms on strings, trees,*

and sequences : computer science and computational biology. Cambridge University Press, 1997.