IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 14, 2019, accepted September 3, 2019, date of publication September 10, 2019,
date of current version September 25, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2940620

Test Case Prioritization Using Firefly Algorithm for Software Testing

MUHAMMAD KHATIBSYARBINI[1], MOHD ADHAM ISA[1], DAYANG N. A. JAWAWI[1], HAZA NUZLY ABDULL HAMED[1], AND MUHAMMAD DHIAUDDIN MOHAMED SUFFIAN[2]

[1]School of Computing, Faculty of Engineering, Universiti Teknologi Malaysia, Johor Bahru 81310, Malaysia
[2]Business Solution and Services, MIMOS Technology Solutions Sdn. Bhd., Kuala Lumpur 57000, Malaysia

Corresponding author: Muhammad Khatibsyarbini (fkmuhammad4@gmail.com)

This work was supported in part by the Fundamental Research Grant Scheme, vote number 5F069, under the Ministry of Education Malaysia, and in part by the Research University Grant Scheme, vote number 01M55, under Universiti Teknologi Malaysia.

ABSTRACT Software testing is a vital and complex part of the software development life cycle. Optimization of software testing is still a major challenge, as prioritization of test cases remains unsatisfactory in terms of Average Percentage of Faults Detected (APFD) and time execution performance. This is attributed to a large search space to find an optimal ordering of test cases. In this paper, we have proposed an approach to prioritize test cases optimally using Firefly Algorithm. To optimize the ordering of test cases, we applied Firefly Algorithm with fitness function defined using a similarity distance model. Experiments were carried on three benchmark programs with test suites extracted from Software-artifact Infrastructure Repository (SIR). Our Test Case Prioritization (TCP) technique using Firefly Algorithm with similarity distance model demonstrated better if not equal in terms of APFD and time execution performance compared to existing works. Overall APFD results indicate that Firefly Algorithm is a promising competitor in TCP applications.

INDEX TERMS Firefly Algorithm, metaheuristic, software engineering, artificial intelligence, search-based software testing, test case prioritization.

I. INTRODUCTION

In the software development process, software testing has a long execution time and can be the most expensive phase [1]. Software testing is arguably the least understood part of the software development process. Even software testing is executed repeatedly, it is often carried out in haste, due to time constraints and fixed resources. In light of this, it has been reported that Test Case Prioritization (TCP) application appears to enhance test viability in software testing activity [2]–[5]. Test case prioritization approach was first mentioned by [6]. That work however only applied prioritization on test cases that had gone through test case selection. Two authors, Rothermel and Harold proposed and evaluated the approach in a further broad context. Considering a test suite as described in Table 1 [7]. Note that this study example depicts an ideal situation in which fault detection information is known.

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Afzal.

TABLE 1. Test suite example.

Test Case	Fault revealed by test case									
	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
TC1	√			√	√			√	√	√
TC2	√				√	√	√			
TC3					√	√	√	√	√	√
TC4	√	√	√		√			√	√	
TC5	√	√	√	√	√					

TCP aims to order a set of test cases to achieve early optimization based on preferred properties [2], [8]. The goal of prioritization is to maximize early fault detection. It is clear that by ordering the test case as TC5-TC3 is far superior than any other combination, as TC5-TC3 order detects all of the faults at an earlier rate as can be seen on Table 1. In real world problems, it is often difficult to determine which tests will actually reveal faults. Hence, the test case prioritization steps depend on several methods, expecting that an early intensification of a certain chosen method will end in boosting of earlier fault discovery. There are many dimensions of the test case prioritization approach. As many as eight broad

dimensions h described by [9]. Those TCP techniques were based on their commonalities in selection procedures, input type, and output type.

TCP can be achieved by utilizing string metrics which are able to distinguish test cases in terms of character differences, followed by prioritization meeting a predefined fitness function. In recent years, existing works appear to have prioritized test cases using only information related to generated test cases [5], [10]–[12]. For instance, by relying on available information such as test case input, software testers may prioritize test cases prior to the availability of system source code. This reduces overall testing time. TCP can be initiated right after the design phase. Among the numerous TCP techniques proposed, there remains room for improvement in terms of time execution performance [4], [5]. This problem is not only widespread on non-artificial intelligence applications in TCP, but also on most Artificial Intelligence (AI) applications in TCP, necessitating improvements especially in terms of APFD and execution time, as indicated by recent studies [3]–[5], [13], [14].

Most AI applications in TCP begin with identifying and differentiating test cases first before prioritization is carried out [10], [15], [16]. A strategy which can be used to differentiate test cases is calculating character differences via string metrics. Once character differences among test cases are established, prioritization can be pursued. String metrics play an important role in textual similarity research. Applications in tasks such as information retrieval, text classification, document clustering, topic detection, topic tracking, questions generation, question answering, essay scoring, short answer scoring, machine translation, and text summarization all utilize string metrics [17]. String metrics can be further categorized based on their metric calculation strategies. In calculating the distance between test cases, specificity and reliability of string distances in the prevailing prioritization goal are needed. In existing works in [3], [5], [10] only one string distance was utilized, which may not resolve redundancy issue where test cases have widespread equidistant weights. This problem may also lead to lowered measures of fault detection at the end of the TCP process.

Upon completion of character difference calculation among test cases using string metrics, an effective prioritization algorithm is then needed to arrange test cases based on respective character difference weights among test cases. Recent works by [3], [18]–[20] indicate that a Swarm Intelligence (SI) algorithm can have a significant effect on TCP process. Additionally, as indicated by findings in [4], the use of artificial intelligence algorithms may yield better APFD results as well as improved execution time performances. However, the uses of SI algorithm such as Firefly Algorithm was not being explored in TCP context which can benefit with the utilization of string metrics.

From the recent studies we can see that there were several problems that arise which worth exploring. To have an increment of APFD result as well as the time execution time, an AI technique is suggested and also recent study shows SI algorithm could have a significant result. Consequently, macro research question or problem of this paper is, ''Which SI algorithm would perform better in TCP context''. Since, Firefly Algorithm was not being explored in TCP, authors has decided to implement the algorithm.

Work in [19] explains the implementation of Firefly Algorithm in generating test cases. The work manages to outperform Ant Colony Optimization (ACO) algorithm. The goal of test case generation is distinguishable from TCP whereby, in test case generation, prioritization of test cases is initiated first, followed by selection of only required test cases. Meanwhile, TCP only involves prioritization of test cases, focusing on ordering of test cases, according to a predefined goal. The work suggests the application of Firefly Algorithm on relevant software testing procedures by means of optimizing the best possible test sequences. Motivated by the suggestion, the schematic of firefly schematic algorithm implemented in the work is possible to be extracted and revised for application in TCP context. Therefore, the research objectives of this paper were.

- To explore prioritizing an optimal ordering of test cases by using Firefly Algorithm, which has not been reported in existing TCP literature.
- To apply the Firefly Algorithm to four different benchmark programs aided with string metrics.
- To compare performance in terms of APFD and execution time performance against existing works.

The rest of the paper is structured as follows. Section 2 describes related works in TCP using artificial intelligence algorithms with string distances. In Section 3, an overview of Firefly Algorithm is outlined. In Section 4, the application of Firefly Algorithm in TCP is presented and illustrated. In Section 5, an empirical evaluation of the proposed work in comparison with existing works is, performed with results and discussion included. Lastly, Section 6 provides a conclusion.

II. RELATED WORK

Regression testing is performed to confirm that alterations have not impacted previously functioning software [21], [22]. As software evolves, test suites have a tendency to grow in size, eventually requiring costly resources to execute. Despite the costs, software testing remains a crucial stage to be executed in order to ensure products built are exactly as intended. Therefore, the principle objective of performing regression test is to guarantee that any alterations acted upon a software system will not influence the unaltered components. When a program has been modified, it is necessary to ensure that any altered parts do not adversely affect unaltered parts of the software, such as the functionalities of that particular software version. In such context, test cases constructed for that software version may only yield accurate outputs for that particular version alone. Those test cases may yield incorrect outputs for subsequent versions of the software created through alterations performed on initial software version. Here, regression testing may prove advantageous.

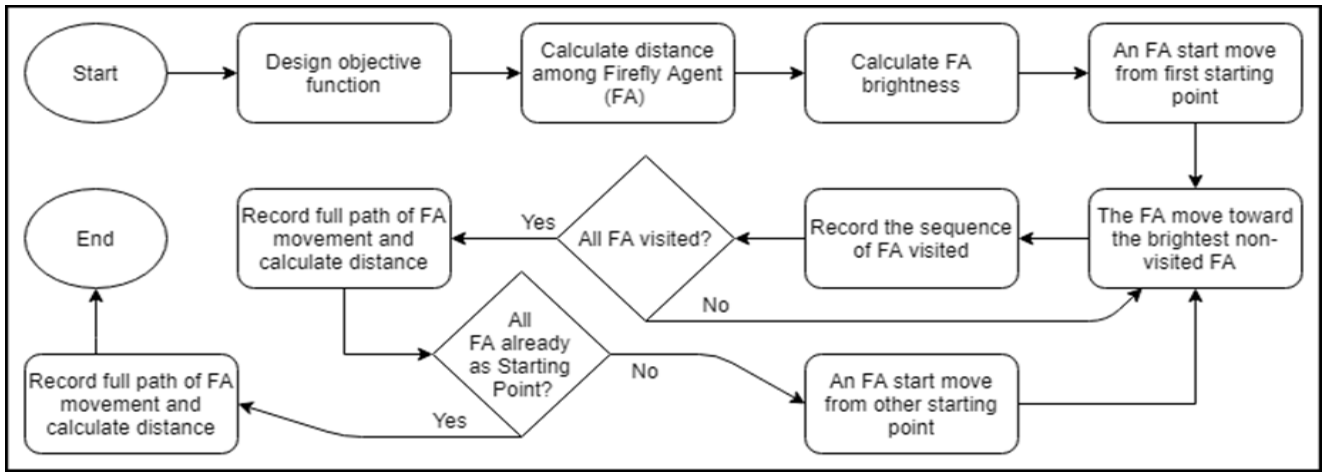

FIGURE 1. Schematic view of firefly algorithm.

During a regression test, changed parts in the software are checked using a regression test technique, in order to check if the unchanged parts of the software behave functionally similar with the original version of the software. Past research indicated that regression testing is an expensive process which may consume more than 33% of the total cost of the software [23]. In [24], various approaches are examined to augment the importance of accumulated test suites in regression testing. A large and complex software system will typically involve a huge pool of test cases to cover all functionalities.

Within regression testing, TCP operates by prioritizing test cases through certain means utilizing only available information. Work in [5] tried to maximize diversity among test cases utilizing test case input information. This is in contrast with work in [10], in which each test case is assigned a distance weight in connection to other test cases, utilizing a string metric such as edit distance, which computes character difference among test cases. These weighted test cases are subsequently prioritized using a predefined algorithm. Regardless of the strategies used, the main goal is fixed to give higher priorities to test cases that are largely dissimilar (i.e., invoke different methods and possess large numerical values of character difference), consequently maximizes test case diversity, which in turn casts a high possibility in unique fault detection [25].

The implementation of artificial intelligence in TCP is not limited to any specific strategy [4]. Within TCP itself, there several algorithms have been used including Genetic Algorithm (GA) [26]–[29], [31]–[34], Greedy, [35], [36], Particle Swarm Optimization [37], [18], and others [5], [13], [38]–[40]. Recent work by Jiang and Chan [5] has indicated that a heuristic prioritization algorithm can have a significant effect on the TCP process. In their finding, the use of artificial intelligence algorithm may increase the yield of average percentage of faults detected (APFD) measure. However, the work did not compare their proposed prioritization algorithm against existing works that have utilized AI algorithms. A comparison study between the proposed prioritization algorithm and existing AI algorithms in TCP would be beneficial to researchers and real world applications, as such study would produce comparative performances in terms of APFD and execution time. Motivated by this, any efforts directed toward fulfilling such beneficial insights which in turn may enhance TCP capabilities are worth exploring.

In [41], nearest neighbor algorithm was used. The approach aimed to reduce the distance between neighboring test cases in the execution order. Distance was quantified as the number of different instructions executed between two test cases. The results from the experiment showed that the order of test case execution instructions is crucial toward yielding a better APFD result. However, the work reported that prioritization execution time utilizing nearest neighbor algorithm yielded the worst performance time. Further, the authors suggest an exploration of possible existing alternative algorithms in TCP, which might yield better performance with respect to program and test suite size.

The Swarm Intelligence (SI) algorithm application in [18] has been reported to offer an overall performance improvement using PSO in a multi-objective TCP. Further, the work reports that PSO prioritization was able to minimize total test cases required to achieve 100% coverage at a shorter execution time. Meanwhile, in [37], PSO implementation in TCP indicated an improvement in APFD result as compared to random ordering. Additionally, the study reports the ability of PSO in TCP process toward improving fault detection capability in the early stages of testing. However, the approach provided limited discussions concerning PSO performance in comparison to other SI algorithms in TCP.

III. OVERVIEW OF FIREFLY ALGORITHM

Light flashes from a firefly draw nearby fireflies. The light flashes can be formulated through associating them with an objective function to be optimized, which in turn makes it possible to formulate a new optimization algorithm. Figure 1 shows a schematic view of the Firefly Algorithm. There are

three assumptions in the Firefly Algorithm discussed in [19] as follows:

a. All fireflies are unisexual. Every firefly either attracts or gets attracted to each of other fireflies.
   - All fireflies will get attracted to any fireflies without any discrimination.
   - For example, there are five fireflies and each of them will get attracted to every single firefly available as they approach one another.

b. The attractiveness of a firefly is directly proportional to the brightness of the firefly.
   - As fireflies are attracted to others available, the brightness of the firefly becomes a priority among them to rank attractiveness.

c. Fireflies move randomly if they do not find a more attractive firefly in adjacent regions.
   - If there are two or more fireflies with same brightness, firefly will random randomly move toward either one.

FA has been applied in spatial fields consisting of different dimensions with promising efficiency and superiority over other algorithms [19], [20]. FA is a metaheuristic algorithm, which assumes that a solution of an optimization problem is encoded as the location of an agent/firefly, while the objective function is encoded as light intensity. Within the Firefly Algorithm are two crucial considerations: the variant of light intensity, sometimes stated as brightness, and formulation of the interaction among fireflies. For simplicity, it is assumed that the spectacle of a firefly is decided by way of its brightness, which in turn is associated with an encoded goal characteristic. Simply put, the attractiveness of a firefly in a search space is directly proportional to the objective function value of the firefly.

The schematic view of the Firefly Algorithm to be applied in TCP, with considerations of assumptions described earlier, is shown in Figure 1. First, the algorithm starts with an objective function derivation. Next the calculation of the adjacency of distance matrix between firefly agent (FA) and its brightness which is encoded to determine the attractiveness of each firefly. Subsequent movement of a firefly will be based on brightness value. The movement stops once all fireflies have been visited. All movements are recorded. Finally, the best sequence of fireflies is chosen based on shortest distance.

IV. APPLYING FIREFLY IN TCP

In the previous section, a brief overview of the basic Firefly Algorithm concept, motivation of Firefly Algorithm in software testing, and a schematic view of the algorithm in TCP context were provided. For clarity, in this section, the flow of Firefly Algorithm application will be described in detail. Table 2 show a firefly itself represents a test case in TCP. The objective function that defines a firefly attractiveness represents a test case similarity weight and uniqueness. All movements of the fireflies are subsequently recorded. The best path is consequently designated the best test suite sequence.

TABLE 2. Firefly algorithm component mapping.

Firefly Component	TCP Component	Justification
Firefly Agent (FA1, FA2, …)	Test Case (TC1, TC2, …)	FA represents a TC. Movement path of FA to other FAs will be recorded as test case ordering in TCP
Firefly Attractiveness (Light and Distance)	Test Case Similarity and Dissimilarity Weights and Uniqueness	FA movement is based on attractiveness which is equivalent to TC similarity weight and uniqueness
Distances Between Firefly	Test Case Distance	FA brightness is decrease based on its travelled distances

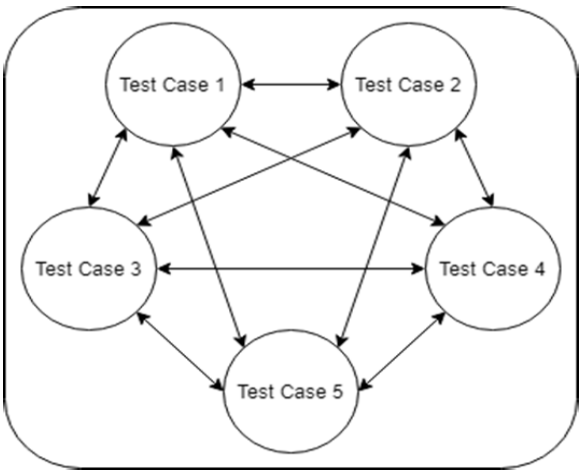

FIGURE 2. Schematic view of test case distance adjacency.

Based on Table 2, the representation of firefly components is illustrated in Figure 2 below. Giving there are five test cases to be prioritized, the probability of having the best prioritized arrangement is one over five factorials (1/5!). From Figure 2, for each test case will act as firefly agent while the distance between each test cases denotes the attractiveness function among firefly agent. To find the best-prioritized arrangement, the Firefly Algorithm is used in this experiment with the string metric as a fitness function.

In this paper, the actual result of distance calculated for each benchmark program will not be illustrated, as the number of test cases and their respective contents are too large and too long to be presented adequately. Instead, to demonstrate how distance is calculated in this research work, five dummy test cases were created. Table 3 shows five dummy test cases created to demonstrate distance calculation and its weight using ''Term Frequency–Inverse Document Frequency'' (TD-IDF). For the distance among test cases, Edit distance metric is used.

Edit distance involves the calculation of string characters, where, the quantified value represents the least number of insertions, deletions, and replacements to transform first string into second string. Edit distance is fixed to involve only a minimal transformation of first string into second string. For example, the edit distance between "9555" and "85577"

TABLE 3. Five dummy test cases.

Test Case	Test Case Content
T1	aaaaa
T2	aaaab
T3	aaabb
T4	aabbb
T5	abbbb

TABLE 4. Adjacency matrix for distance between test cases.

	T1	T2	T3	T4	T5
T1	0	0.80	0.60	0.40	0.20
T2	0.80	0	0.80	0.60	0.40
T3	0.60	0.80	0	0.80	0.60
T4	0.40	0.60	0.80	0	0.80
T5	0.20	0.40	0.60	0.80	0

is 3, since three edits are needed to transform "9555" into "85577". The following edits illustrate edit distance calculation:

1) **9**555 → **8**555 (substitution of "8" for "9");
2) 855**5** → 855**7** (substitution of "7" for "5");
3) 8557 → 8557**7** (insertion of "7" at the end).

In TFIDF, term frequency, *tf*, calculation starts first with the use of selected term or string frequency in a document. Simply put, *tf* refers to the amount of term t occurring within a document d. Meanwhile, the formula for inverse document frequency, *idf*, deals with the significance of a term in a pool of documents. Subsequently, TFIDF equation is expressed as follows.

$$\text{TFIDF(t)} = t/T \times \log^{N}/n_t$$

where: t = term frequency in one document; T = term frequency in all documents; N = number of term occurrence throughout document; and n_t = number of documents has term t. These two string metrics were then used to calculate the distance and weight of the dummy test cases in Table 3 below.

The results of calculated distance for dummy test cases in Table 3 are shown in Table 4. All string distances were quantified based on percentage of similarity from scale 0 to 1. Value 1 denotes the highest differences of two test cases while, value 0 denotes zero distance of two test cases. For the test case weight which will represent as the brightness of each test cases, the value of all test cases weight will be similar among each other using TD-IDF formulation as all have different single string. TF-IDF is a standout scheme amongst most typically utilized term weighing schemes in data retrieval methods. Attributed to this capability, TF-IDF has been regularly applied in experimental investigations [42]. The weight among test cases was used to assign brightness among firefly agents in Firefly Algorithm. The value would be calculated as the product of one multiple by one with log five over one which will have the value of 0.6989. Therefore, to make it easier to demonstrate the calculation within the Firefly Algorithm, we assume the weight of all test cases which act as the brightness of firefly agent as 1. Table 5 show the test cases initial point and distance travelled in every movement.

TABLE 5. Test cases movement and update.

Movement Update		Starting Point				
		TC1	TC2	TC3	TC4	TC5
1st Move	Test Case	TC5	TC5	TC1	TC1	TC1
	Distance Traveled	0.20	0.40	0.60	0.40	0.20
2nd Move	Test Case	TC2	TC1	TC5	TC5	TC4
	Distance Traveled	0.60	0.60	0.80	0.60	0.60
3rd Move	Test Case	TC4	TC4	TC2	TC2	TC2
	Distance Traveled	1.20	1.00	1.20	1.00	1.20
4th Move	Test Case	TC3	TC3	TC4	TC3	TC3
	Distance Traveled	2.00	1.80	1.80	1.80	2.00

From Table 5, the selection of the next test cases in movement update were based on the weight of the test case which act as the brightness over the distance between the test cases, where the highest will be selected as the next move. In mathematical formulation, the brightness of a test cases over distance calculation can be denoting as:

$$\frac{\textit{Weight of current Test Case}}{\textit{Distance to Next Test Case}}$$

Since the weight of all dummy test cases were all the same, shorter distance were the only metric considered. Based on Table 5, four starting points of movement end up with TC3 as the last point to visited, and the lowest distanced travelled is 1.80 which tied with three best different sequences. This is due to the similar weight of all dummy test cases. However, in a real case study or benchmark program, the similar weight problem diminishes and distances would be more dynamic. To summarize the implementation of Firefly Algorithm in test case prioritization, Figure 3 illustrates the flowchart of the Firefly Algorithm used.

Based on Figure 3, the data were parsed from benchmark programs and test case were extracted from the dataset. Then calculation of test cases distances was performed by using edit distance and with using TFIDF string metric. The test case adjacency was then generating a matrix table as similar to the dummy adjacency matrix in Table 4. Prioritization then started with the first movement of an FA toward the brightest non-visited test case until it reached the last firefly in the search space. The overall path travelled would represent a prioritized test case ordering. The shortest distance of full sequence of the test cases is considered as the best path.

V. EMPIRICAL SETUP

In experiment setup phase, the experiment context, variables, datasets, and design to validity are defined as follows.

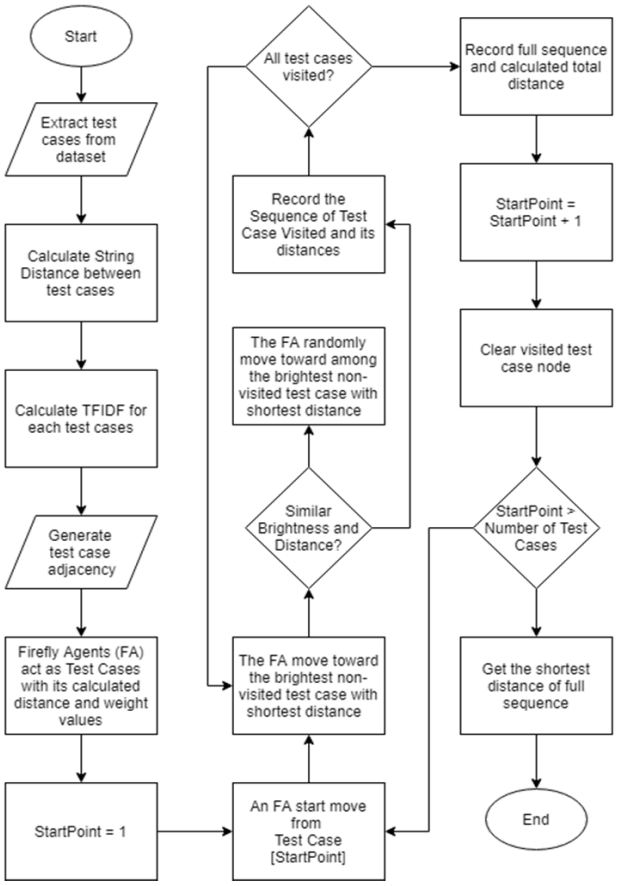

FIGURE 3. Flowchart of applied firefly algorithm.

A. EXPERIMENT CONTEXT

The experiment was performed in a controlled laboratory setting. Two machines were used to run their experiment, the first one is a Linux-based machine, run on a computer with 4GB RAM and Core i5 Intel processor. This first machine was used to extract and execute test cases from each of the three benchmark Siemens TCAS benchmark dataset on a Linux environment. The second machine, equipped with Intel i7 and 16 GB RAM, is used to run permutation of test cases using four string metrics, implemented in Java NetBeans 8.0.2 and measure experiment prioritization timing performance, which required higher processing power as the test case dimensionality matrix is large, with the test suite size standing at 1052×1052 cells.

B. EXPERIMENT VARIABLES

The experiment variables comprise of dependent variable and controlled variable. The dependent variable is an observed element which is affected by changes made to the controlled variables. Controlled variable refers to elements that the authors alter to induce behaviors that are observable. The two sets of variables are as follows:

Controlled variables:

- String metrics: Edit Distance and TFIDF
- Siemens TCAS Benchmark Program
- Real-world UNIX Benchmarks Programs

TABLE 6. Benchmark suite.

Benchmark Program	Description	Line of Code	No. Faulty Versions	Test Pool Size
flex	Lexical Analyzer	8571 - 10124	21	567
grep	Text Searcher	8053 - 9089	17	809
gzip	File Compressor	4081 - 9289	55	217
tcas	Aircraft Avoidance System	4081 - 9289	41	1027
cs-tcas	Improved Aircraft System	4681 - 11289	42	1608
j-tcas	Aircraft Avoidance System	5681 - 15289	42	2108

Dependent Variables:

- Average Percentage of Faults Detected (APFD)
- Time Execution

For benchmark applications, three real-world UNIX benchmarks programs and three SIEMENs benchmarks programs were used as test suites (downloaded from http://sir.unl.edu) [43]. These benchmark programs are popular among researchers [5] in search-based software testing, especially in TCP. In order to assess the overall performance and consistency of the proposed applied algorithm in TCP process, we ran the prioritization process of each algorithm with 30 iterations for every benchmark program. The 30 iterations were intentionally being chosen to the consistency of each algorithm since every single iteration were made might resulted in slightly different results. Table 6 shows the details of benchmark applications chosen for this study.

From Table 6, 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 comes 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. Simple example of fault matrix can be referred from Table 1. For TCAS program, the test cases do not lie in any folder. The test case was written in a test script itself. Figure 4 shows the main function for TCAS C program, while Figure 5 shows the extracted test case from the test script.

As can be seen in figure 4, there were 12 variables declared which gain its value from retrieved argument from the inputs. Therefore, in Figure 5, there were also 12 columns where each column contains the value input for each variable in TCAS main function. The TCAS program were then used for the experiment process design. Each of our benchmark program accepts a file as an input and a string of inputs entered from a command line terminal. Input data from each file along with its respective command line content was mined, converted it into strings, and calculated for its similarity distance and uniqueness by using an edit distance. Based on the distance calculated, the test case finally will be prioritized. Table 7 lists existing algorithms which

```
main(argc, argv)
int argc;
char *argv[];
{
    if(argc < 13)
    {
    fprintf(stdout, "Error: Command line arguments are\n");
    fprintf(stdout, "Cur_Vertical_Sep, High_Confidence, Two_of_Three_Reports_Valid\n");
    fprintf(stdout, "Own_Tracked_Alt, Own_Tracked_Alt_Rate, Other_Tracked_Alt\n");
    fprintf(stdout, "Alt_Layer_Value, Up_Separation, Down_Separation\n");
    fprintf(stdout, "Other_RAC, Other_Capability, Climb_Inhibit\n");
    exit(1);
    }
    initialize();
    Cur_Vertical_Sep = atoi(argv[1]);
    High_Confidence = atoi(argv[2]);
    Two_of_Three_Reports_Valid = atoi(argv[3]);
    Own_Tracked_Alt = atoi(argv[4]);
    Own_Tracked_Alt_Rate = atoi(argv[5]);
    Other_Tracked_Alt = atoi(argv[6]);
    Alt_Layer_Value = atoi(argv[7]);
    Up_Separation = atoi(argv[8]);
    Down_Separation = atoi(argv[9]);
    Other_RAC = atoi(argv[10]);
    Other_Capability = atoi(argv[11]);
    Climb_Inhibit = atoi(argv[12]);

    fprintf(stdout, "%d\n", alt_sep_test());
    exit(0);
}
```

FIGURE 4. Main function of TCAS C Program.

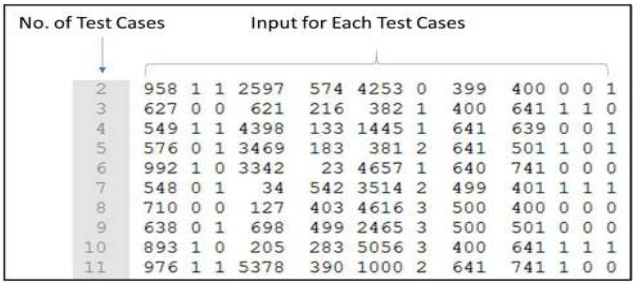

FIGURE 5. Example inputs for each test cases for TCAS C program.

TABLE 7. Prioritization techniques.

Prioritization Technique	Brief Description
PSO [3], [19]	Iteratively improves search for a solution.
LBS [5]	Best promising case is prioritized first in a limited set.
GA [35]	A natural genetic selection based on an adaptive search.
Greedy [8], [44]	Biggest/largest coverage search.
Firefly	Brightest search for each iteration.

were used to compare their prioritization performances against our proposed Firefly Algorithm.

From Table 7, Particle Swarm Optimization (PSO) is chosen as one of comparable algorithm because, it is a quite similar swarm intelligence algorithm. Apart from that, PSO also has been widely used in TCP area. Local Beam Search (LBS) on the other hand is one of latest algorithm implemented in TCP and has quite convincing result. While for Genetic Algorithm (GA) and Greedy, both of them were chosen as they were the common technique used in TCP apart from random. Table 8, summarize the reasoning behind the selection of selected algorithm for this experiment.

C. EXPERIMENT FLOW

Figure 6 illustrates an overview of TCP experiment flow. The flow is divided into four phases; (1) Information Extraction phase, (2) String Distance and TFIDF calculation for

TABLE 8. Selected algorithm comparative analysis.

Prioritization Algorithm	Study Aim	Summary of Findings
Greedy [41]	To minimize the distance between neighboring test cases in the execution order.	The results showed better APFD result. However, the drawback is the execution time were at the worst.
GA [35]	A simulation to study test case prioritization permutation	The results were a good result. However, execution time were at the worst as the permutation increase
LBS [5]	An evaluation on beam search, greedy, optimal, hill climbing and genetic algorithm.	Beam search significantly improve the execution time. Beam search algorithm is more effective than the other four prioritization algorithm.
PSO [3], [18]	Evaluation PSO with multi objective purposed.	PSO prioritization able to minimize the total test case required achieving 100% coverage
	To have higher early fault detection rate.	Improvement in APFD result. Improve the capability of fault detection in earlier

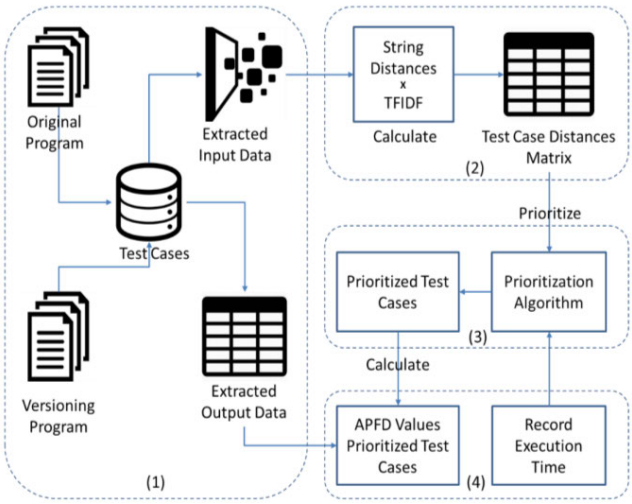

FIGURE 6. Overview of experiment flow.

fireflies' distances and brightness, (3) Prioritization phase, and (4) Evaluation phase.

In phase 1, test cases were extracted from repository with their input contents collected. Extracted input content for each test case was placed on a separate text document to ease calculation process in the next phase. Then, the original program and versioning programs were compiled and executed. Outputs produced from the original program were compared against versioning programs. Test cases which revealed faults, indicated by test cases which yielded different outputs when ran on the original program and when ran on versioning programs, were recorded. The record was kept in the form of a fault matrix sheet.

In phase 2, character difference between test cases was calculated based on extracted input contents. Computed weights were subsequently populated into a test case distance matrix. The matrix placed similarity and dissimilarity weights and uniqueness of test cases in their respective positions.

TABLE 9. Fault matrix for distance between test cases.

Test Case	Fault Number				
	F1	F2	F3	F4	F5
TC1		✓	✓	✓	
TC2	✓	✓	✓		
TC3			✓		✓
TC4	✓			✓	✓
TC5			✓	✓	✓

The weights and uniqueness are represented the firefly attractiveness values in the proposed Firefly Algorithm.

In phase 3, the proposed Firefly Algorithm was executed to prioritize test suites, where the shortest possible cumulative test case distance was selected as the best test case ordering, which reflects the best efficiency in terms of execution time. A complete flow of the proposed Firefly Algorithm application in TCP is illustrated in Figure 3. The flowchart covers extraction phase until prioritization phase. The best path travelled by fireflies represents the best sequence of prioritized test cases. Finally, in phase 4, prioritized test suites were evaluated by calculating their APFD measures based on fault matrix of their respective programs, obtained in phase 1. Time execution for each prioritization technique was also recorded to evaluate time execution efficiency.

D. EVALUATION METRIC

Average percentage fault detection rate (APFD) metric measurement was used to quantify the rate of fault detection [8]. APFD has been widely used in numerous TCP experiments. APFD is a metric used to quantify how rapid an optimized ordering of test cases can discover faults [2], [7]. The results of APFD values in this study ranged between zero to 100, where a greater value indicates a better fault revealing rate. The equation for calculating APFD value is shown as follows:

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

where T is a test suite containing n test cases and 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. The APFD value of T' is calculated by using the APFD equation. To demonstrate the calculation of APFD, one of the best sequences from dummy Test Cases in Table 5 is used. The sequence selected is TC2 – TC5 – TC1 – TC4 – TC3. Giving that the fault matrix from extracted output as in Table 9 below.

Based on the Table 9 and APFD equation, the mathematical calculation for the APFD would be as follow:

$$APFD = 1 - \frac{1+1+1+2+2}{5 \times 5} + \frac{1}{2(5)}$$

Therefore, the value of APFD for the prioritized sequence for dummy dataset is 0.82.

VI. RESULT AND DISCUSSION

The experiments began with calculation of string distances using Edit distance for all three benchmark applications.

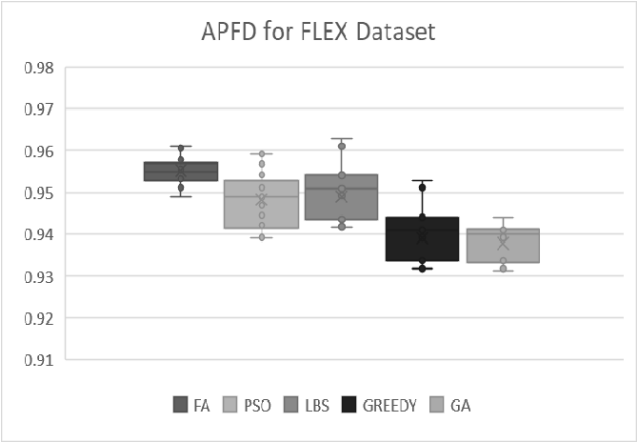

FIGURE 7. APFD for *FLEX* program.

TABLE 10. Overall APFD assessments for *GZIP*.

	FA	PSO	LBS	GREEDY	GA
Mean	**0.9553**	0.9483	0.9490	0.9390	0.9377
Median	**0.9550**	0.9489	0.9510	0.9410	0.9401
Min	**0.9489**	0.9392	0.9418	0.9318	0.9312
Max	0.9611	0.9591	**0.9629**	0.9529	0.9441
Std-Dev	**0.0032**	0.0065	0.0060	0.0060	0.0044

Upon completion of string distance calculation, Firefly Algorithm was applied. Prioritization performance of the proposed Firefly Algorithm was subsequently compared with PSO, GA, Local Beam Search (LBS) and Greedy.

A. THE APFD RESULTS

In this section, APFD results for each benchmark program is illustrated using boxplots as shown in Figure 7. Overall APFD assessment for each benchmark program is also tabulated as presented in Table 9. All five benchmark programs are characteristically distinguishable in terms of input data despite stark differences in terms of program sizes. The discussion of Firefly Algorithm in comparison to other algorithms are provided at the end of this section.

Figure 7 shows a boxplot for APFD results while Table 10 shows overall APFD assessment for *FLEX* benchmark program. Cross-referring the table and boxplot, it can be concluded that Firefly Algorithm shows a better APFD result in *FLEX* test application. The result shows Firefly Algorithm is slightly better compared to other compared algorithms, it still shows superior mean, median, and standard deviation values. However, for max values, LBS somehow manage to get the highest values with 0.9629. Even so, all other values already indicate that the Firefly Algorithm is able to produce consistently higher APFD signifying its applicability in *FLEX* benchmark program context.

Figure 8 shows a boxplot for APFD results while Table 11 shows the overall APFD assessment for *GREP* benchmark program. Quite similar to the previous dataset,

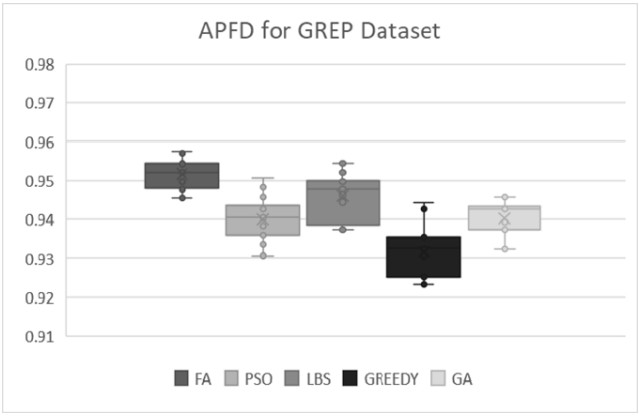

FIGURE 8. APFD for *GREP* program.

TABLE 11. Overall APFD assessments for *GREP*.

	FA	PSO	LBS	GREEDY	GA
Mean	**0.9517**	0.9401	0.9461	0.9317	0.9403
Median	**0.9520**	0.9404	0.9479	0.9326	0.9427
Min	**0.9454**	0.9307	0.9374	0.9233	0.9324
Max	**0.9576**	0.9506	0.9544	0.9444	0.9457
Std-Dev	**0.0034**	0.0055	0.0056	0.0059	0.0045

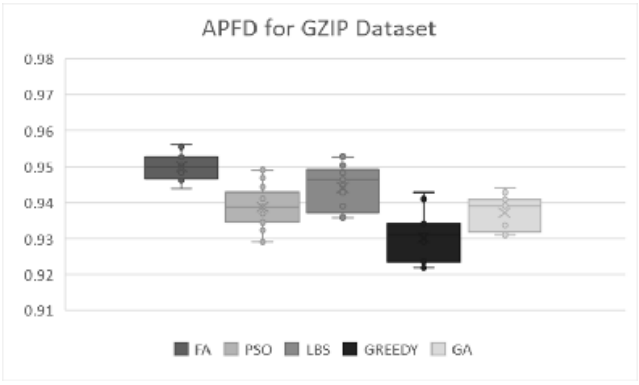

FIGURE 9. APFD for *GZIP* program.

TABLE 12. Overall APFD assessments for *GZIP*.

	FA	PSO	LBS	GREEDY	GA
Mean	**0.9501**	0.9389	0.9442	0.9303	0.9372
Median	**0.9500**	0.9389	0.9464	0.9310	0.9392
Min	**0.9439**	0.9292	0.9359	0.9218	0.9311
Max	**0.9561**	0.9491	0.9529	0.9429	0.9441
Std-Dev	**0.0032**	0.0057	0.0059	0.0063	0.0049

by cross-referring the table and boxplot, it can be concluded that Firefly offers a better APFD result in *GREP* application.

Figure 9 shows a boxplot for APFD results, while Table 12 shows the overall APFD assessment for *GZIP* benchmark program. Again, similar to the previous dataset, by cross-referring the table and boxplot, it can be concluded that Firefly offers a better APFD result in *GZIP* application. Overall Firefly performed better in all UNIX

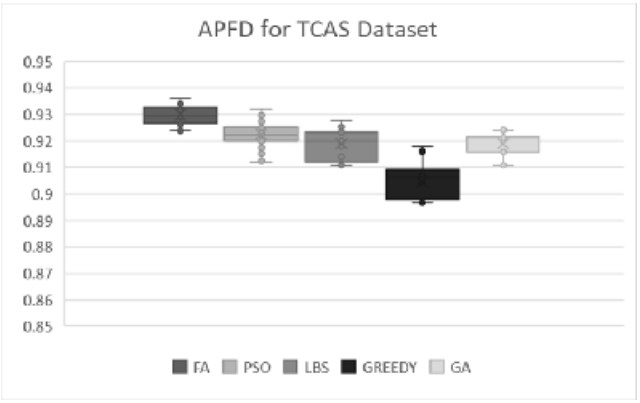

FIGURE 10. APFD for *TCAS* program.

TABLE 13. Overall APFD assessments for *TCAS*.

	FA	PSO	LBS	GREEDY	GA
Mean	**0.9298**	0.9224	0.9188	0.9044	0.9189
Median	**0.9294**	0.9223	0.9200	0.9059	0.9211
Min	**0.9238**	0.9121	0.9108	0.8967	0.9108
Max	**0.9360**	0.9320	0.9278	0.9178	0.9241
Std-Dev	**0.0034**	0.0050	0.0056	0.0063	0.0044

TABLE 14. Overall APFD assessments for *CS-TCAS*.

	FA	PSO	LBS	GREEDY	GA
Mean	0.9297	0.9256	0.9187	0.9160	0.9196
Median	0.9294	0.9261	0.9215	0.9190	0.9211
Min	0.9238	0.9151	0.9108	0.9041	0.9108
Max	0.9354	0.9340	0.9278	0.9231	0.9267
Std-Dev	0.0031	0.0050	0.0059	0.0066	0.0049

Benchmarks Programs over 30 iterations. As for Siemens program, Figure 10 and Table 13 articulate the results obtained.

From Figure 10 and Table 13, once again a similar convincing result from Firefly Algorithm is obtained. However, in *TCAS* program, it is quite interesting to see that almost all algorithms tend to have a nearly consistent value for each iteration. Even so, by cross-referring the table and boxplot, it can be concluded that Firefly shows a better APFD result in *TCAS* application.

From Figure 11 and Table 14, once again a similar convincing result from Firefly Algorithm is obtained. However, CS-*TCAS* starts to show inconsistent results while GA has more consistency based on the std-dev value. Even so, by cross-referring the table and boxplot, it can be concluded that Firefly shows a better APFD result in CS-*TCAS* application.

As for final J-TCAS, Figure 12 and Table 15, once again show a quite similar result for the Firefly Algorithm overcoming the other algorithms. However, similar to CS-TCAS, J-*TCAS* which is the largest program, the std-dev values show an increment also since both of them are large dataset.

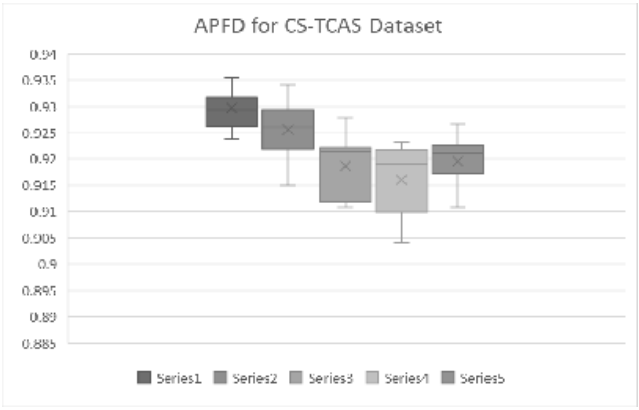

FIGURE 11. APFD for CS-*TCAS* program.

TABLE 15. Overall APFD assessments for *J-TCAS*.

	FA	PSO	LBS	GREEDY	GA
Mean	0.9275	0.9257	0.9181	0.9192	0.9196
Median	0.9263	0.9261	0.9206	0.9214	0.9211
Min	0.9206	0.9151	0.9070	0.9060	0.9108
Max	0.9354	0.9340	0.9267	0.9267	0.9267
Std-Dev	0.0044	0.0052	0.0058	0.0054	0.0049

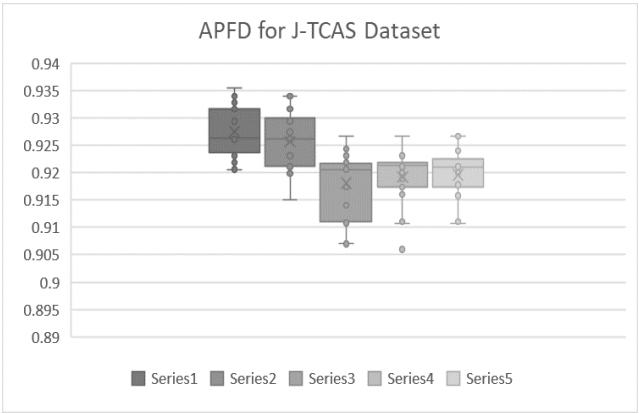

FIGURE 12. APFD for *J-TCAS* program.

Even so, it is still acceptable, and Firefly shows a better APFD result in J-*TCAS* application. In order to see the consistency of APFD values, Figures 13 -18 show the graphs of all iteration values of APFD for each dataset.

From Figure 13-18, Firefly Algorithm indicates a higher consistency and suitability to be applied in TCP domain as APFDs value obtained at each iteration, across three programs, did not show any sharp fluctuation. Any sharp fluctuation is unfavorable as this may indicate a "by chance occurrence". Firefly strategy has the best overall APFD result as the selection of subsequent firefly path is solely driven by next brightest firefly agent. In cases of more than one agent with equal brightness, firefly path will randomly select a firefly agent. LBS on the other hand looks promising but the algorithm hardly depends on a constant width value where different constant value set will have a significant impact

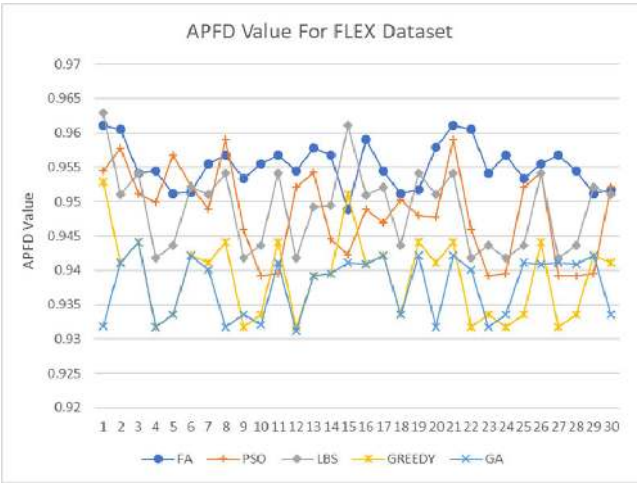

FIGURE 13. APFD graph for *FLEX* benchmark program.

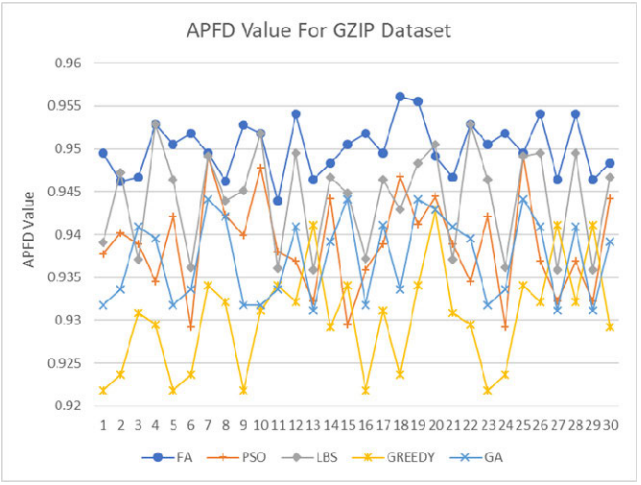

FIGURE 14. APFD graph for *GZIP* benchmark program.

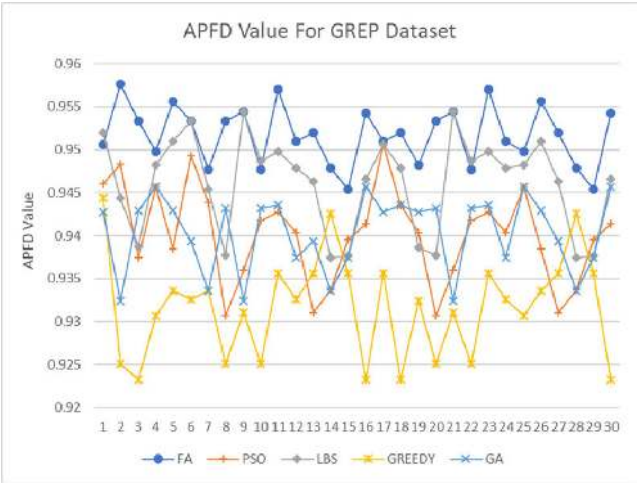

FIGURE 15. APFD Graph for *GREP* benchmark program.

on APFD results. Meanwhile, GA exhibits a unique behavior attributed to its adaptive search, which may vary prioritization APFD score, as the performance highly depends on its mutation fitness function. For Greedy algorithm, the

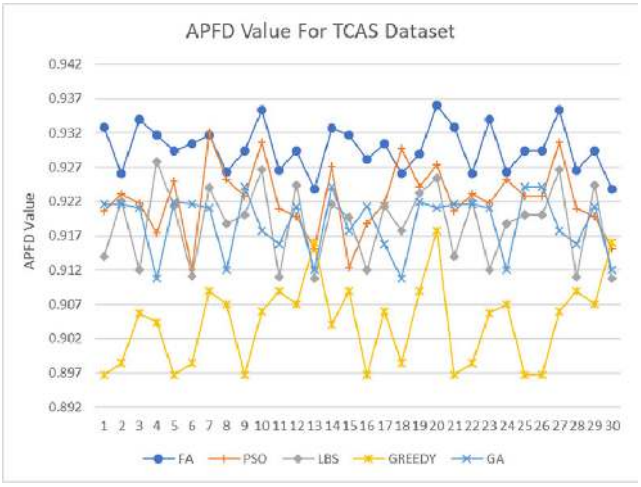

FIGURE 16. APFD graph for *TCAS* benchmark program.

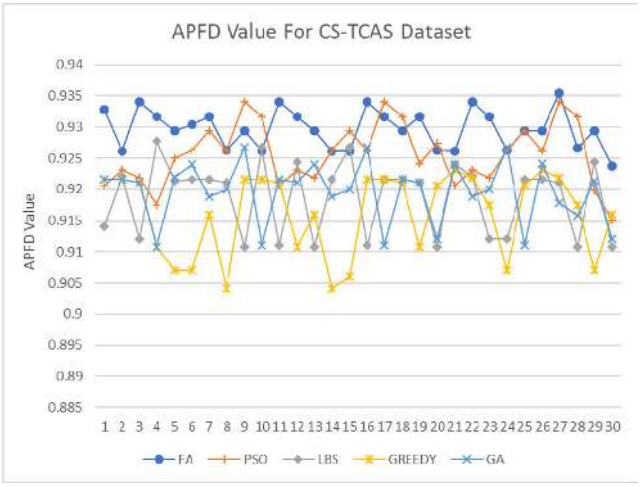

FIGURE 17. APFD graph for *CSTCAS* benchmark program.

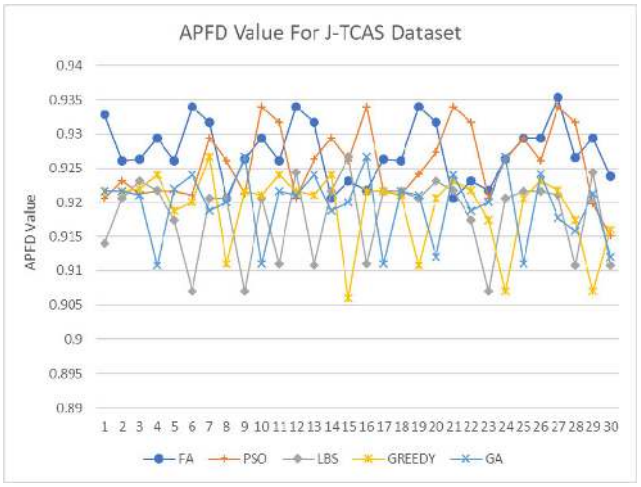

FIGURE 18. APFD graph for *JTCAS* benchmark program.

algorithm searches for next agent which covers the furthest distance which sometimes produces a sub-optimal solution path. Overall, firefly strategy has the best overall APFD result as the selection of the next path is solely driven by next brightest agent. The strategy also optimizes ordering of test cases, by promoting test case diversity, as test cases with least similarity are chosen first, which in turn may provide a higher probability in detecting faults earlier.

TABLE 16. Average execution time for benchmark programs.

Algorithm	Average Time Execution (sec)					
	GREP	GZIP	FLEX	TCAS	CS-TCAS	J-TCAS
PSO	343	300	320	800	1200	1450
LBS	228	**180**	**210**	**400**	**600**	**750**
GA	1856	1255	1520	3560	4550	5240
Greedy	240	220	230	460	650	800
Firefly	**220**	185	**210**	**390**	**580**	**650**

B. THE EXECUTION TIME RESULTS

In terms of algorithm efficiency, Table 16 show the average time taken for all prioritization algorithms to complete searching for solution paths in all five benchmark programs. Firefly Algorithm, if not equal, outperforms LBS in terms of average time taken to prioritize test cases in *flex* benchmark program, while greedy comes as second best. In *gzip* benchmark, LBS strategy outperforms other algorithms while firefly comes as the second fastest prioritizing algorithm. In *grep*, firefly outperforms all prioritization algorithms, with execution time 220 seconds, outperforming LBS by eight seconds. For J-*tcas* the biggest benchmark program which the biggest program among all four program used, Firefly have the fastest time execution with 650, 100 seconds faster than LBS. This result may be attributed to firefly strategy, which solely focuses on next brightest firefly agent in searching, resulting in a reduced search time. In addition, the number of iterations were also fixed based on the sizes of benchmark programs.

C. THE OVERALL RESULTS AND DISCUSSION

Table 17 summarizes overall results of prioritizing algorithms across three benchmark programs. From Table 13, for *grep* benchmark program, firefly outperforms other search strategies, yielding the highest mean of APFD value (0.9517). LBS comes as second best strategy while PSO comes at third. APFD values among these three search strategies exhibit slight differences, as all of them are inspired from one common search strategy, which is best first search strategy. Firefly strategy for selecting next path solely depends on next brightest agent. When there is one or more agents with equal brightness, an agent will be randomly selected, resulting in distinguishable best path in every iteration. Meanwhile, LBS algorithm hardly depends on a fixed constant width value, where different constant values set subsequently produce different results. Highest APFD mean values for *gzip* and *flex* programs in the experiments were also yielded by Firefly Algorithm with values, 0.9501 and 0.9553, respectively while for *tcas* benchmark program is 0.9298.

TABLE 17. Summary OF APFD and time performance.

Algorithm	Benchmark Program							
	grep		*gzip*		*flex*		*tcas*	
	Mean APFD	Time	Mean APFD	Time	Mean APFD	Time	Mean APFD	Time
PSO	0.9401	343	0.9389	300	0.9483	320	0.9224	800
LBS	0.9461	228	0.9442	**180**	0.9490	**210**	0.9188	400
GA	0.9403	1856	0.9372	1255	0.9377	1520	0.9189	3560
Greedy	0.9317	240	0.9303	220	0.9390	230	0.9044	460
Firefly	**0.9517**	**220**	**0.9501**	185	**0.9553**	**210**	**0.9298**	**390**

For overall time results, size of test cases does affect the execution efficiency of prioritization process. The greater is the number of test cases, the longer is the time taken to complete an entire prioritization process. For *tcas* benchmark program, which has the largest number of test cases, Firefly strategy scored the best prioritization execution time, 390 seconds in average, while LBS recorded 400 seconds, which is the second fastest. In *flex* benchmark program, Firefly and LBS recorded an equal execution time, 210 seconds. However, in *gzip* program, LBS outperforms other algorithms (180 seconds), five seconds much faster than Firefly in prioritizing test cases. This result could be attributed to the behavior of LBS algorithm inspired by best search first concept which possesses a superior execution time. Refined attractiveness function that was used to determine firefly brightness has managed to reduce the count of agents with equal brightness. In turn, this has reduced the number of random selection of agents when there are more agents with equal brightness. Consequently, this reduced overall execution time for the Firefly in prioritizing test cases. For LBS, the difference in brightness values does have an influence on execution time. LBS algorithm hardly depends on a constant value for beam width, which inevitably has affected the overall time execution. GA on the other hand, exhibited a unique behavior with its adaptive search, where each solution path produced is distinct attributed to its mutation fitness function at a considerably longer execution time. For Greedy algorithm, the algorithm searches for agents which possess highest brightness first, which may yield a sub-optimal solution path. Overall, Firefly Algorithm performs better in prioritizing small to medium size test suites. FA performance across three benchmark programs did not indicate stark differences in terms of average execution time, indicating its capability in yielding approximately acceptable solution paths with a certain extent of consistency. In addition, FA also produces better APFD measures indicating its suitability to be used as a competitive prioritizing algorithm in TCP research area.

VII. THREAT OF VALIDITY

This paper has some limitation acknowledged which may possibly threaten its legitimacy. The potential threat of this applied soft computing paper is associated with less diversity in selection of benchmark program and incomplete data extraction.

A. SELECTION OF DATASET

In benchmark program selected in this paper were only four where three are from Real-world UNIX Benchmarks Programs and three from Siemens Suites where all can be obtained from http://sir.unl.edu [43]. From Table 5, the limitation of Real-world UNIX Benchmarks Programs is, all of them have small test cases pool. Possible issues may arise if the proposed technique applied bigger sizer of test cases pool. To reduce this problem, TCAS program were used as it has large test cases pool. Other than selection of benchmark program, authors does not apply the algorithm into the real cases study which might have different kind of test cases structure which may have significant affect towards the results.

B. INCOMPLETE DATA EXTRACTION

In this experiment, the data extracted from the test cases were only the input data and also the output data. The value for distances and weight of a test cases were solely based on input data of each test cases while the output is used as the fault matrix for APFD calculation. There are a lot of other data were not fully utilized in this experiment such as the system coding itself, the instruction within the test cases, the steps and others. This incomplete data extraction could be the next potential threat to this paper. To reduce this problem, instead of using one calculation metric, authors used two type of string metric which is string distance metric and weight metric to assign the attractiveness to each firefly agent accurately.

C. INCOMPLETE DATA EXTRACTION

In this paper, the experiment was solely focused on test-case based TCP technique. Other techniques which have quite similar strategies with this technique were not included in our validation. It would be unfair if we compared different base of TCP technique to assess which algorithm is better. To reduce this problem, we had run the experiment for 30 times for each algorithm to see the consistency of the result obtained. However, it might not be enough to see which algorithm actually fits better in which TCP techniques. This is an opportunity for future work to explore.

VIII. CONCLUSION

This paper used a Firefly strategy in prioritizing test cases to achieve higher APFD results. Three benchmark programs were used to validate the efficiency and effectiveness of Firefly Algorithm in prioritizing test suites, in comparison with other prioritization algorithms. The experimental results showed that Firefly obtained the highest APFD scores compared to other prioritization algorithms. The experiment also showed that Firefly Algorithm slightly outperformed LBS in terms of execution time. Overall, the APFD results showed

that Firefly Algorithm may become a strong competitor in the TCP area. The APFD results indicate that Firefly Algorithm could be effective in discovering fault proneness issues, which is strongly required in safety critical systems. In future work, it would be interesting to explore possible enhancement on this particular swarm intelligence algorithm focusing on coverage efficiency.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to members of Embedded & Real- Time Software Engineering Laboratory (EReTSEL), Software Engineering Research Group (SERG), Faculty of Engineering, UTM and MIMOS for their feedback and continuous support.

REFERENCES

[1] G. J. Myers, T. M. Thomas, and C. Sandler, *The Art of Software Testing*, vol. 1. Hoboken, NJ, USA: Wiley, 2004.
[2] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: An empirical study," in *Proc. IEEE Int. Conf. Softw. Maintenance (ICSM)*, Aug./Sep. 1999, pp. 179–188.
[3] M. Khatibsyarbini, M. A. Isa, and D. N. A. Jawawi, "A hybrid weight-based and string distances using particle swarm optimization for prioritizing test cases," *J. Theor. Appl. Inf. Technol.*, vol. 95, no. 12, pp. 2723–2732, 2017.
[4] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Inf. Softw. Technol.*, vol. 93, pp. 74–93, Jan. 2018.
[5] B. Jiang and W. K. Chan, "Input-based adaptive randomized test case prioritization: A local beam search approach," *J. Syst. Softw.*, vol. 105, pp. 91–106, Jul. 2015.
[6] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal, "A study of effective regression testing in practice," in *Proc. 8th Int. Symp. Softw. Rel. Eng.*, Nov. 1997, pp. 264–274.
[7] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Trans. Softw. Eng.*, vol. 27, no. 10, pp. 929–948, Oct. 2001.
[8] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, pp. 159–182, Feb. 2002.
[9] Y. Singh, "Systematic literature review on regression test prioritization techniques," *Informatica*, vol. 36, pp. 379–408, Dec. 2012.
[10] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, "Prioritizing test cases with string distances," *Autom. Softw. Eng.*, vol. 19, no. 1, pp. 65–95, 2012.
[11] L. Mei, W. K. Chan, T. H. Tse, B. Jiang, and K. Zhai, "Preemptive regression testingof workflow-based Web services," *IEEE Trans. Services Comput.*, vol. 8, no. 5, pp. 740–754, Sep./Oct. 2015.
[12] S. W. Thomas, H. Hemmati, A. Hassan, and D. Blostein, "Static test case prioritization using topic models," *Softw. Eng.*, vol. 19, no. 1, pp. 182–212, 2014.
[13] D. Gao, X. Guo, and L. Zhao, "Test case prioritization for regression testing based on ant colony optimization," in *Proc. IEEE Int. Conf. Softw. Eng. Service Sci.*, Sep. 2015, pp. 275–279.
[14] R. M. Parizi, A. Kasem, and A. Abdullah, "Towards gamification in software traceability: Between test and code artifacts," in *Proc. 10th ICSOFT*, Jul. 2015, pp. 1–8.
[15] A. Shahbazi and J. Miller, "Black-box string test case generation through a multi-objective optimization," *IEEE Trans. Softw. Eng.*, vol. 42, no. 4, pp. 361–378, Apr. 2016.
[16] C. Catal and D. Mishra, "Test case prioritization: A systematic mapping study," *Softw. Qual. J.*, vol. 21, no. 3, pp. 445–478, 2013.
[17] W. H. Gomaa and A. A. Fahmy, "A survey of text similarity approaches," *Int. J. Comput.*, vol. 68, no. 13, pp. 13–18, 2013.
[18] M. Tyagi and S. Malhotra, "Test case prioritization using multi objective particle swarm optimizer," in *Proc. Int. Conf. Signal Propag. Comput. Technol. (ICSPCT)*, Jul. 2014, pp. 390–395.
[19] P. R. Srivatsava, B. Mallikarjun, and X. S. Yang, "Optimal test sequence generation using firefly algorithm," *Swarm Evol. Comput.*, vol. 8, pp. 44–53, Feb. 2013.
[20] N. Iqbal, K. Zafar, and W. Zyad, "Multi-objective optimization of test sequence generation using multi-objective firefly algorithm (MOFA)," in *Proc. Int. Conf. Robot. Emerg. Allied Technol. Eng. (iCREATE)*, Apr. 2014, pp. 214–220.
[21] H. K. N. Leung and L. White, "Insights into regression testing," in *Proc. Int. Conf. Softw. Maintenance*, Oct. 1989, pp. 60–69.
[22] S. Elbaum, G. Rothermel, and J. Penix, "Techniques for improving regression testing in continuous integration development environments," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*, 2014, pp. 235–245.
[23] P. K. Chittimalli and M. J. Harrold, "Recomputing coverage information to assist regression testing," *IEEE Trans. Softw. Eng.*, vol. 35, no. 4, pp. 452–469, Jul. 2009.
[24] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Softw., Test. Verification Rel.*, vol. 22, no. 2, pp. 67–120, 2012.
[25] H. Hemmati, A. Arcuri, and L. Briand, "Empirical investigation of the effects of test suite properties on similarity-based test case selection," in *Proc. 4th IEEE Int. Conf. Softw. Test., Verification Validation*, Mar. 2011, pp. 327–336.
[26] R. Maheswari and D. Mala, "Combined genetic and simulated annealing approach for test case prioritization," *Indian J. Sci. Technol.*, vol. 8, no. 35, pp. 1–5, 2015.
[27] Y. Lou, D. Hao, and L. Zhang, "Mutation-based test-case prioritization in software evolution," in *Proc. IEEE 26th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2015, pp. 46–57.
[28] F. Yuan, Y. Bian, Z. Li, and R. Zhao, "Epistatic genetic algorithm for test case prioritization," in *Proc. Int. Symp. Search Based Softw. Eng.*, 2015, pp. 109–124.
[29] C. Catal, "On the application of genetic algorithms for test case prioritization: A systematic literature review," in *Proc. 2nd Int. Workshop Evidential Assessment Softw. Technol.*, 2012, pp. 9–14.
[30] A. Kaur and S. Goyal, "A genetic algorithm for fault based regression test case prioritization," *Int. J. Comput. Appl.*, vol. 32, no. 8, pp. 975–8887, 2011.
[31] W. Jun, Z. Yan, and J. Chen, "Test case prioritization technique based on genetic algorithm," in *Proc. Int. Conf. Internet Comput. Inf. Services*, Sep. 2011, pp. 173–175.
[32] S. Sabharwal, R. Sibal, and C. Sharma, "Prioritization of test case scenarios derived from activity diagram using genetic algorithm," in *Proc. Int. Conf. Comput. Commun. Technol. (ICCCT)*, Sep. 2010, pp. 481–485.
[33] K. Deb, S. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[34] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," *IEEE Trans. Softw. Eng.*, vol. 32, no. 9, pp. 733–752, Sep. 2006.
[35] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Trans. Softw. Eng.*, vol. 33, no. 4, pp. 225–237, Apr. 2007.
[36] S. Li, N. Bian, Z. Chen, D. You, and Y. He, "A simulation study on some search algorithms for regression test case prioritization," in *Proc. 10th Int. Conf. Qual. Softw.*, Jul. 2010, pp. 72–81.
[37] A. K. Joseph, G. Radhamani, and V. Kallimani, " Improving test efficiency through multiple criteria coverage based test case prioritization using Modified heuristic algorithm," in *Proc. 3rd Int. Conf. Comput. Inf. Sci. (ICCOINS)*, Aug. 2016, pp. 430–435.
[38] S. Eghbali and L. Tahvildari, "Test case prioritization using lexicographical ordering," *IEEE Trans. Softw. Eng.*, vol. 42, no. 12, pp. 1178–1195, Dec. 2016.
[39] K. Solanki, Y. Singh, S. Dalal, and P. R. Srivastava, "Test case prioritization: An approach based on modified ant colony optimization," in *Emerging Research in Computing, Information, Communication and Applications*. 2016.
[40] T. Noguchi, H. Washizaki, Y. Fukazawa, A. Sato, and K. Ota, "History-based test case prioritization for black box testing using ant colony optimization," in *Proc. IEEE 8th Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2015, pp. 1–2.
[41] P. Stratis and A. Rajan, "Test case permutation to improve execution time," in *Proc. 31st IEEE/ACM Int. Conf. Automat. Softw. Eng. (ASE)*, 2016, pp. 45–50.

[42] A. Aizawa, "An information-theoretic perspective of tf–idf measures," *Inf. Process. Manage.*, vol. 39, no. 1, pp. 45–65, Jan. 2003.
[43] *Software-Artifact Infrastructure Repository: Home*. [Online]. Available: http://sir.unl.edu/portal/index.php
[44] L. Zhang, D. Hao, L. Zhang, G. Rothermel, and H. Mei, "Bridging the gap between the total and additional test-case prioritization strategies," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 192–201.

MUHAMMAD KHATIBSYARBINI was born in Selama, Perak, Malaysia. He received the B.S. degree in software engineering from the Universiti Teknologi Malaysia, Johor, Malaysia, in 2016, and the M.S. degree in master of philosophy from the Universiti Teknologi Malaysia, in 2018. During his study, he was a Research Assistant (2016–2018) with the Embedded and Real-Time Software Engineering Laboratory (EReTSEL) and Software Engineering Research Group (SERG). Also, during those two years, he has been working as a contract Software Engineer with two different companies: Mechabotic Enterprise and Effron Sdn Bhd. His research interests include software engineering, search-based software testing, and artificial intelligence.

MOHD ADHAM ISA received the bachelor's degree in computer science from Universiti Teknologi Malaysia, the master's degree in computer science, and the Ph.D. degree in software engineering from Universiti Teknologi Malaysia (UTM), Malaysia, where he is currently the Head of the Software Engineering Research Group (SERG). His main research interests include software engineering, software quality, software testing, requirement engineering, and software project management. A major part of his research projects focuses on software quality assurance, real-time embedded systems, as well as the Internet of Things (IoT).

DAYANG N. A. JAWAWI received the bachelor's degree in software engineering from Sheffield Hallam University, U.K., and the master's degree in computer science and the Ph.D. degree in software engineering from Universiti Teknologi Malaysia (UTM), Malaysia, where she is currently an Associate Professor with the School of Computing, Faculty of Engineering. Her main research interests include software engineering, software reuse, software quality, software testing, requirement engineering, and computing education. A major part of her research projects focuses on rehabilitation and mobile robotics, real-time embedded systems, as well as precision farming applications.

HAZA NUZLY ABDULL HAMED received the first degree in information technology, majoring in artificial intelligence, from Universiti Utara Malaysia, the master's degree in computer science from Universiti Teknologi Malaysia, and the Ph.D. degree from the Auckland University of Technology, New Zealand. He is currently a Senior Lecturer with the School of Computing and a founding member of the Applied Industrial Analytics Research Group (ALIAS), Faculty of Engineering, Universiti Teknologi Malaysia (UTM). Before joining UTM, he worked as a Web Programmer and System Analyst. His research interests are computational intelligence, evolutionary computation, deep learning, optimization, spatiotemporal data processing, and information system development.

MUHAMMAD DHIAUDDIN MOHAMED SUFFIAN received the B.Tech. (Hons.) degree in business information systems and the M.Sc. degree in computer science – real-time software engineering from Universiti Teknologi PETRONAS and Universiti Teknologi Malaysia, respectively. He is currently the Senior Staff Engineer with the Business Solutions and Services Division, MIMOS Technology Solutions Sdn. Bhd, a subsidiary of MIMOS Berhad, the National Applied R&D Centre. He is a certified Six Sigma Green Belt, a Certified Tester Advanced Level-Test Manager (CTAL-TM), and a Certified Tester Foundation Level (CTFL). His experiences encompass various spectrums of ICT across various sectors, including government, automotive, banking, and education. Currently, he assumes the role of Lead Business System Analyst for an ICT Court Transformation Project. His research interests focus on software requirement, software testing, software quality, and software process. He holds two patents filed under his name, of which one as main inventor and another one as co-inventor.