SEMARAK ILMU PUBLISHING

Journal of Advanced Research in Applied Sciences and Engineering Technology

Journal homepage:
https://semarakilmu.com.my/journals/index.php/applied_sciences_eng_tech/index
ISSN: 2462-1943

An Enhancement of Multi-Factor Weighted Approach Technique in Prioritizing Test Cases by Comparing Similarity Distance

Alaa Alrhman Mohammed Raweh Al-Shaibani[1], Johanna Ahmad[1,*], Rohayanti Hassan[1], Salmi Baharom[2], Dwinta Suci Antari[3]

[1] Department of Software Engineering, Faculty of Computing, Universiti Teknologi Malaysia, 81310 Johor Bahru, Johor, Malaysia
[2] Faculty of Computer Science and Technology, Universiti Putra Malaysia, 43400 Serdang, Selangor, Malaysia
[3] Universitas Internasional Batam, Kota Batam, Kepulauan Riau 29426, Indonesia

ARTICLE INFO

Article history:
Received 22 June 2023
Received in revised form 17 October 2023
Accepted 24 July 2024
Available online 10 August 2024

Keywords:

Mutation testing; Test case prioritization; Distance; Software testing; Jester mutation tool

ABSTRACT

Software testing is one of the most critical phases in the software development life cycle model (SDLC), where the quality of a software product is evaluated. Test case prioritization (TCP) is used to prioritize and schedule test case execution to conduct higher-priority test cases to optimize the software testing process. Traditionally, techniques rely on source code or a specification for the tested system. Therefore, numerous factors and techniques have been used to optimize the prioritization process. One of the factors is distance. String Distance aims to find the degree of similarity between the test cases, which helps prioritize the test case according to the dissimilar value. The higher the dissimilarity value, the higher the probability of detecting new faults. Previous research has used Jaccard Distance to measure the distance to prioritize test cases with the same priority value. In the meantime, the Manhattan Distance is used in this research as it provides a better measure of distance. Our aim of this research is to compare and evaluate both Jaccard and Manhattan Distance algorithms in terms of their effectiveness to formulate the enhancement of the previous multi-factor weighted Approach. The research experiment has shown the process of calculating the Distance matrix for the sample Java Programs and subsequent evaluation using the mutation testing approach and APFD calculation. The results of The Average Percentage of Fault Detected (APFD) of the Test case prioritization by the Manhattan Distance matrix have obtained a higher value, validating its hypothesized effectiveness.

1. Introduction

Software Testing is one of the most important steps in the Software Development Life Cycle (SDLC). Over the years, various testing techniques and strategies have been proposed to improve the efficacy of defecting faults during testing processes that are limited by resources, cost, and timeliness [1]. One of the techniques used to address the concerns during the testing process is test case prioritization (TCP). Many tests case prioritization (TCP) techniques occur to prioritize test cases

* *Corresponding author.*
E-mail address: johanna@utm.my

https://doi.org/10.37934/araset.50.1.238249

based on the number of covered methods, previous version execution history, requirement dependencies, location-based testing, and the order in which test cases are executed, depending on their importance.

The number of TCP approaches prioritize the execution of test cases based on their importance, implying that they are inadequate at discovering software flaws [6]. According to [2], the complexity, redundancy, frequency, permutation, fault matrix, and distance are the factors that significantly affect the TCP techniques' efficacy and efficiency. As a result, a weighted strategy was offered as an effective predictor for determining the best test case sequencing and priority based on the weightage of each test case calculated throughout the prioritization process [2]. Due to time constraints, only Jaccard Distance has been employed in the MFWA technique to prioritize the test cases that received the same weightage after considering all (MFWA) factors.

Moreover, the issue of handling the same priority value was not handled efficiently in many studies as they are executed in the same order in which they occur, or they might be executed randomly, which subsequently introduces a new problem when many test cases are given the same priority value. This research aims to enhance the MFWA technique for test case prioritization by applying the Manhattan and Jaccard distance to formulate and imply the enhancement. The proposed algorithms are evaluated to determine their effectiveness in test case prioritization. This research aims to enhance the MFWA technique by comparing similar distance prioritization algorithms, such as Manhattan distance and Jaccard distance, to identify the best techniques for distance-based prioritization applications and a new Distance algorithm, which can increase software testing efficacy and performance by discovering defects.

2. Methodology

This research's methodology framework consists of four phases:

i. Literature Review
ii. Problem Definition
iii. Experiment and evaluation
iv. Result Dissemination.

Figure 1 illustrates the research framework.

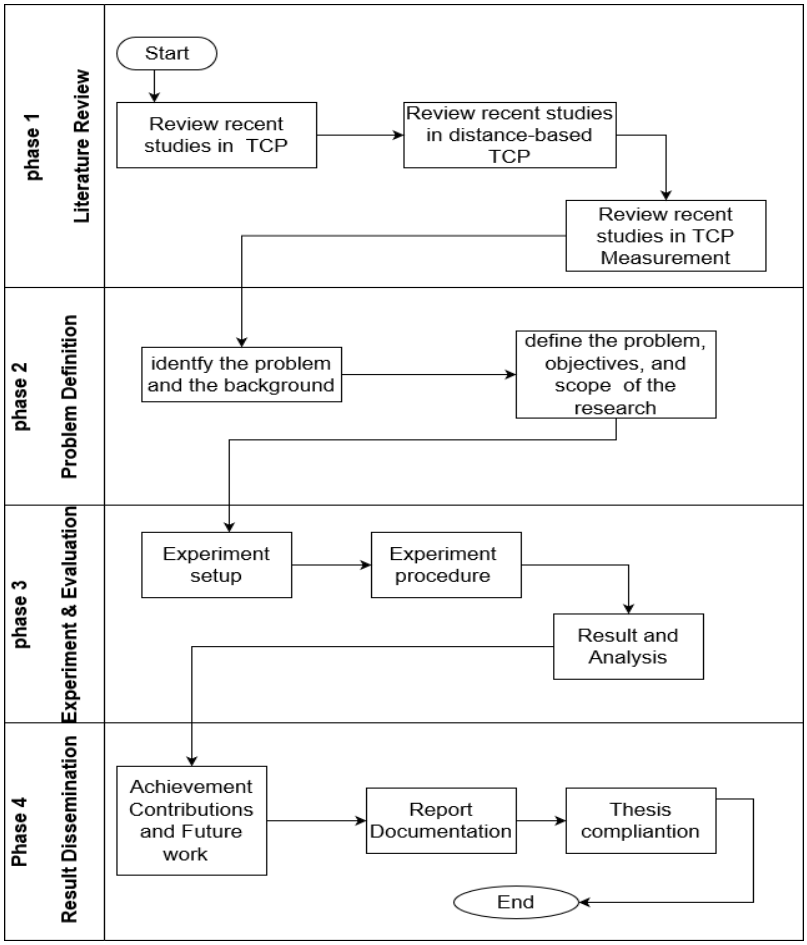

Fig. 1. Research Methodology

3. Experimental Setup

In this experiment, several mutants were generated for the user programs. 20 mutants were introduced for the Circular Queue program, and 28 mutants were generated for the Bank program. Mutation testing, as defined, is a technique of software testing that includes altering a program's source code in small parts [3]. According to [18], mutation testing simulates genuine problems by seeding many errors into the original code through a sequence of mutation operators. Hence, each modified program is referred to as a mutant. Mutation testing has been used and applied by many researchers as it provides a high evaluation of the efficiency of the test suite for detecting faults [2,5,17]. This study believes a test case kills the mutant when it behaves differently in a mutant program from the original one.

The Jester tool has been used to automate the mutation testing process; it is a JUnit test tester that modifies the program code in several ways and verifies whether the tests fail due to each mutation. It can also detect code executed but not tested during the tests.

3.1 Subject Program

This research used the same code programs and test cases as in [1] previous research MFWA. Using the same program and test cases would ensure a more reliable comparison of the TCP of both string distance measures, as both utilize the same programs and test cases. Table 1 illustrates the programs and their code structure.

Table 1
Experiment used programs

Subject Program	Lines of Codes	Number of classes	Number of Methods	Number of Test cases
Circular Queue	69	1	3	32
Bank	257	1	7	40

3.2 Experimental Design

The flowchart in Figure 2 illustrates the experimental design used in this study. The chart was developed following a comprehensive examination of the above experiments. The design of the experiment is represented in 2-layer architecture.

The first layer is the Presentation layer, which marks the beginning of the experiment as it contains the input components, Java programs and the test suites, followed by the second Application layer, which processes the input components. Finally, the output component's results were reviewed and saved continuously.

The Jaccard Distance and Manhattan Distance for each method class of the programs were manually calculated In the Application layer. Then, mutants were generated by installing Jester mutation testing in the program. The techniques' effectiveness was then evaluated using the Average Percentage of Faults Detected (AFPD).

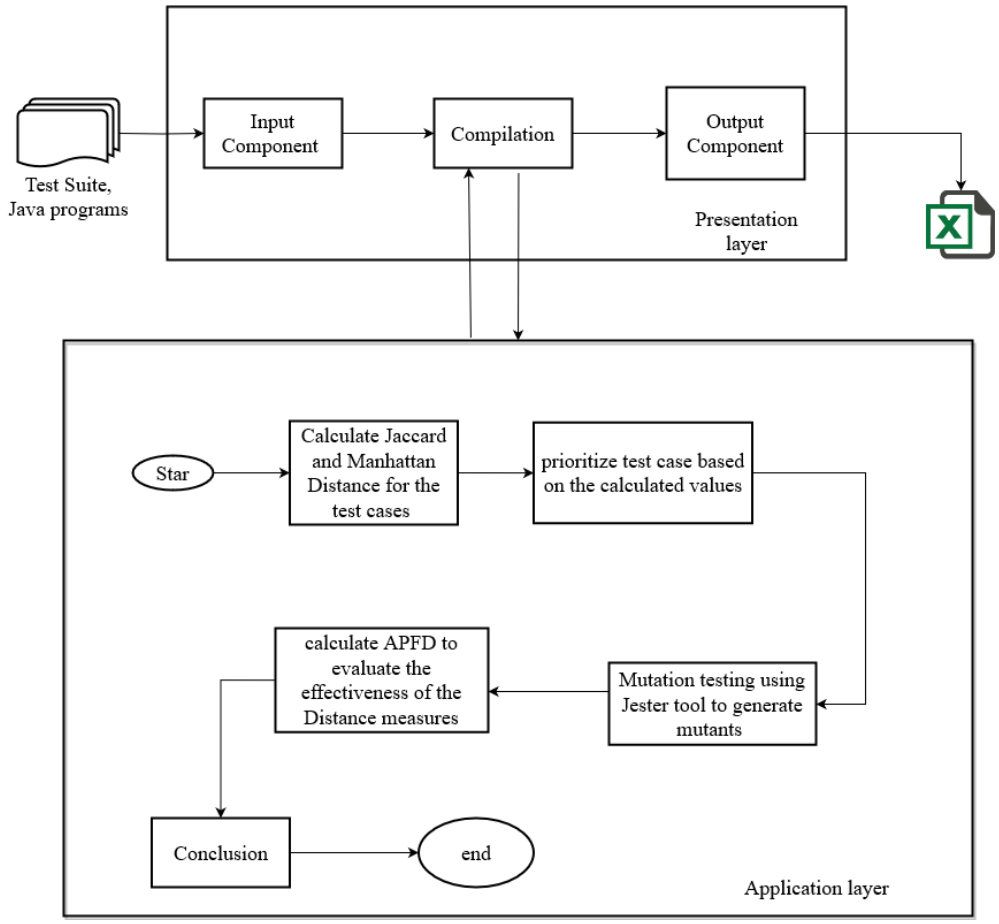

Fig. 2. Experimental Design

Jester provides the option to define the required mutations inside a configuration folder for mutation testing. The mutant generator is responsible for producing either standard or customized

mutants. The tool runs the mutants against the test suite and appropriately calculates the mutant's test score. Table 2 shows the mutant operators selected for the Jester tool and inserted into the program for evaluating mutations.

Table 2
Experiment Mutant Operators

No.	Mutation operators
1	Change numerical constants (Mutate 0 to 1, 1 to 2)
2	Mutate true to false and vice versa
3	Mutate if (condition) to if (false && condition)
4	Mutate if (condition) to if(true || condition)
5	Mutate ++ to – and vice versa
6	Mutate != to == and vice versa

4. Related Work

4.1 Software Testing

The software development life cycle (SDLC) comprises processes for developing any software product through different development phases. One of these essential phases is testing, ensuring the software product is defects-free and ready to be released and used. Testing determines whether or not a specific system meets its original requirements [14]. Software testing is also performed to ensure the software is reliable, compatible, efficient, and resilient. Testing is costly, but ignoring this phase is even more costly. It is an important part of Software Quality Assurance, and many companies spend up to 40% to 50% of their development time and cost on product testing [8]. Software defects may be expensive or even fatal, as errors in software can result in monetary and human losses. As a result, testing is crucial to prevent software defects from occurring [10]. Testing does not ensure that the system being tested is error-free. It can be used to demonstrate the presence of errors but not to demonstrate their absence [12]. It can only detect flaws or errors that are already known. It does not indicate any problems that have yet to be discovered [15].

4.2 Test Case Prioritization

According to [13], test case prioritization (TCP) aims to identify an ordering of test cases that optimizes the value of a fitness function corresponding to a certain testing objective, such as the number of found defects or code coverage. Increased fault detection rates may offer early feedback on the system under test, enabling faster debugging and enhancing the possibility that, if testing is halted prematurely, only the test cases with the highest fault detection capabilities were performed within the available testing time [5]. In this light, prioritizing test cases is a safer approach regarding defect discovery since it does not delete test cases but permutes them [13]. Test case prioritization is normally applied to differentiate the properties of each test case; if two test cases are similar, one will receive a lower priority during the prioritization process [9]. In the meantime, there are different techniques for prioritization. They typically select test cases according to their structural coverage information, such as statement coverage, branch coverage, and method coverage [19].

4.3 String Distance Algorithms

Different techniques have been introduced for prioritization. These techniques typically select test cases according to their structural coverage information, such as statement coverage, branch

coverage, and method coverage [19]. Distance is one of the potential factors discovered by previous research that can improve the effectiveness of the testing phase [7]. Several existing test case prioritization techniques used string distance algorithms to determine the degree of similarity between test cases and reorder the test cases in the test suite according to their similarity value. Compared to prioritization techniques, studies indicate that using distance prioritization algorithms provides higher efficiency in detecting faults [4,7,9]. However, there is a comparable difference between different string distance algorithms, as previous researchers showed. The research concern is analysing the efficiency and effectiveness of Jaccard Distance and Manhattan Distance Algorithms.

The findings by [13] indicated Manhattan distances as the best choice of test case prioritization based on string distances for the strongest mutants. Each character in a string is compared against another string's character in a character-based string metric. For instance, In the Manhattan distance, a string of length n can be considered an n-dimensional space vector of characters, and each of those characters has an ASCII code (or any other numerical coding). According to [13], the higher the value calculation of the Manhattan distance, the greater the difference between the test cases. Therefore, this test case would be executed first.

Moreover, [1] proposed using distance to provide a unique weight for each test case to solve the issue of finding the same priority value in the same test cases. The Jaccard Distance, also known as the Jaccard Similarity Coefficient, was used in place of it in the study. The Jaccard Distance has been used to assess the similarity and dissimilarity to measure the coverage of program entities between two test cases [11].

5. Result and Discussion

This study compared and evaluated the efficiency of applying Manhattan and Jaccard distances in prioritized test cases receiving the same value or weight. Both programs were used to formulate the enhancement of the MFWA distance factor. Two programs were used in this experiment: the Circular Queue program and The Bank program.

5.1 Distance Algorithms Value Calculation

First, the values of the program's methods were calculated, referring to the complexity value of each method. These values were then used to calculate the test case values of both programs as the preparation step of obtaining distance using Manhattan and Jaccard distance algorithms. For instance, in this preparation step, Circular Queue methods add (), remove (), and front () were given values 3,2,1, respectively, add () can be considered more complex and receives the highest complexity value among the methods. Then, the methods were replaced or substituted on the test cases with their values to get the total value of the test cases. For example, test case 3 in Figure 3 will be after the substitution (3,3,3,2,3,3,3,1) to get the value of the test case. All the method values will be added up as (3+3+3+2+3+3+3+1= 21).

TC 3	_.add(1).add(1).add(1).remove().add(1).add(1).add(1).front()

Fig. 3. Example of test case of Circular Queue Program

5.2 Manhattan Distance Calculation

The Manhattan distance calculation was obtained using its string distance Eq. (1) [13] on the following formula:

$$\sum_{i=1}^{n} |xi - yi| \quad (1)$$

After each method on the test cases was replaced with its values, the researcher developed a string consisting of a series of numbers. These strings were used to calculate the Manhattan distance between the test cases to end up with the Manhattan distance matrix, which involves the distance values from any test case. In other words, the distance value between any test case and all other test cases has been calculated.

Calculating the string distance between the test cases produced a 32*32 matrix for Manhattan Distance for the Circular Queue program since it has 32 test cases. Furthermore, the matrix allows the measurement of the distance between any of the program test cases, so it can be referred to this matrix in any case where test cases having the same value or weight to determine which test case should be executed first, in this case, the test case that has higher distance value from last ordered test case is selected. For example, based on Table 3, which is part of the Manhattan Distance Matrix of the Circular Queue program, if the test case is ordered as TC2, TC3, TC6, TC4, and TC1, TC5 has the same weight, one test case need to be selected to be executed after TC4, so in this case, the Manhattan Distance value should be checked for both test cases from TC4, using the distance matrix that has been produced, it can be found that Distance values are 3 and 37 for TC1, TC5 respectively which means that TC5 will be executed first and followed by TC1 which have lower distance value. The same technique has also been used for the Bank program, which has 40 test cases. Hence, the researcher produced a 40*40 matrix to calculate the Manhattan Distance value between the test cases.

Table 3
Manhattan Distance Matrix Sample

No.	TC1	TC2	TC3	TC4	TC5	TC6
TC1	0	55	20	3	40	17
TC2	55	0	39	52	27	44
TC3	20	39	0	17	22	5
TC4	3	52	17	0	37	14
TC5	40	27	22	37	0	23
TC6	17	44	5	14	23	0

5.3 Jaccard Distance Calculation

The Jaccard distance calculation was obtained using its string distance Eq. (2) [11] on the following:

$$\text{Jaccard Distance } (\rho_a, \rho_b, \rho_a, \rho_b) = 1 - \frac{|\rho_a \cap \rho_b|}{|\rho_a \cup \rho_b|} = 1 - \frac{|\rho_a \cap \rho_b|}{|\rho_a \cup \rho_b|} \quad (2)$$

As shown in the Eq. (2), two main elements must be calculated: the intersection between test cases and the union. So, to find the intersection, there is a need to identify the number of similar elements between the test cases by comparing each element on one test case string and the corresponding element on the other. In this light, the value of 1 was given if both elements were

similar; otherwise, the value given is 0. After completing this comparison process, all the values from comparing the string elements are added to find the total number of intersections between the test cases. Moreover, in the case of having unequal string length, any different value of the longer string can be put to complete the comparison as it will end up having 0 as there is no intersection between those elements. The union can be obtained using two different ways, whether to use the count of the longest test case string between the two compared test cases or first to make all test cases have the same length as the longest test case between all test cases by filling the string with any value that will not affect the process of getting the intersection value. This experiment used the second method to ensure consistency, as the Jaccard distance value between all test cases was calculated. It was used for both programs to produce a similar matrix as the one produced for the Manhattan distance calculation. The Jaccard distance value was calculated using the equation after calculating the intersection and the union values between each and all test cases for Circular Queue and bank programs.

After calculating the Jaccard distance for both programs, the researcher obtained a 32*32 distance matrix for the Circular Queue program for the 32 test cases and a 40*40 matrix for the Bank program for the 40 test cases. These two matrixes keep all the distance values between the test cases, making checking the distance between any test cases easier and more efficient. In any case, it is found that test cases with the same value or weight could be used to determine which test case should be executed first based on the Jaccard distance value in the four similarity categories.

If two or more test cases have the same value or weight, the more dissimilar test case should receive higher priority and be executed first. In other words, the test case with a distance value closer to zero from the last ordered test case must be executed first. Using the same example used on Manhattan distance calculations, if the test cases are ordered as TC2, TC3, TC6, TC4, and TC1, TC5 have the same weight, in this case, when referring to the matrix Table 4, it can be found that TC5 will be executed first as its value is 0.93 that less and much closer to 0 than TC1 value which is 1.

Table 4
Jaccard Distance Matrix Sample

No.	TC1	TC2	TC3	TC4	TC5	TC6
TC1		1	1	1	1	1
TC2	1		0.78	0.96	0.7	0.91
TC3	1	0.78		0.88	0.67	0.5
TC4	1	0.96	0.88		0.93	0.86
TC5	1	0.7	0.67	0.93		0.6
TC6	1	0.91	0.5	0.86	0.6	

5.4 Mutation Testing

Mutation testing was designed to assure the quality of a software testing suite, as it should not leave many lines of code uncovered. Thus, test cases should identify and distinguish the inserted mutations from the original code. In this experiment, mutation testing was carried out using Jester as it automates the testing process more feasibly and efficiently than manual mutation testing, which is considered difficult because of its combination. There were several steps in this phase, from downloading the tool from its official website and choosing the mutation operators for generating the mutants inserted into the programs used to test execution. The selection of mutation operators is critical in mutation testing since ineffective mutants will fail to cause test cases to fail, defeating the goal of mutation testing. Therefore, in this research, the choice of operators was according to Table 2, which shows the operators and the operands of the mutation. The mutations selected for

the used programs depended on the nature of the program as they vary in structure. Moreover, these operators that provide extreme changes are very effective as they are more likely to generate faults and defects, which were found to be a perfect match for this research experiment.

5.5 Calculating the APFD Value of Manhattan Distance

Various measurement techniques were used to measure the effectiveness of the TCP methods of test case prioritization. Researchers have used different measuring methods to evaluate the effectiveness of the TCP. Of these methods, Average Percentage Fault Detected (APFD) was used and calculated using Eq. (3).

$$APFD = 1 - \frac{TF1 + TF2 + \ldots TF_N}{n\ x\ m} + \frac{1}{2\mathrm{n}} \quad (3)$$

Where:
TF = is the position of the first test in the test suite T that i exposes fault *i.*
m= is the total no. of faults exposed in the system or module under T.
n= is the total no. of test cases in T.

The Fault matrix generated by Manhattan Distance for both programs can be found in Appendix E and F. APFD was computed for the Circular Queue program as follows:

n = 32*, m* = 20
APFD = $1 - \frac{337}{32 \times 20} + \frac{1}{2\times32}$ = 0.4890625

Whereas the calculation of APFD for the Bank program is calculated as follows:

n = 40, *m* = 28
APFD = $1 - \frac{76}{40\times28} + \frac{1}{2\times40}$ = 0.944642857

6. Calculating APFD value of Jaccard Distance

Eq. (3) was used for both programs to calculate the APFD of the Jaccard Distance, starting from the Circular Queue program calculation as follows

n = 32, *m* = 20
APFD = $1 - \frac{352}{32\times20} + \frac{1}{2\times32}$ = 0.459375

For the Circular Queue program, the APFD prioritized fault matrix using Jaccard distance values was 0.45, which means the APFD of the prioritized fault matrix using Manhattan distance values was higher at 0.48. This difference in the APFD of both algorithms resulted from the different priority ranks of test cases. This also resulted from the different ways that each distance algorithm has been discussed earlier. Table 5 shows ten test cases with the highest priority value for both metrics to illustrate this point. Note that the distance value was only applied to the test cases that received the same weight; '/' indicates that the distance value was not used on the test case.

Table 5
Priority List for Circular Queue Program

Rank	Manhattan Distance priority list		Jaccard Distance priority list	
	Test case no.	Manhattan Distance value	Test case no.	Jaccard Distance value
1	TC 20	\	TC 20	\
2	TC 27	\	TC 27	\
3	TC 26	12	TC 23	0.16
4	TC 23	9	TC 26	0.3
5	TC 2	5	TC 12	0.5
6	TC 12	4	TC 2	0.8
7	TC 7	\	TC 7	\
8	TC 15	39	TC 15	0.8
9	TC 21	39	TC 21	0.8
10	TC 3	\	TC 3	\

It can be observed that apart from the test cases with "/" on the distance value, the distance values were only used in cases where the researcher obtained the same weight or complicity value and 8th and 9th rank with the same distance values for both algorithms. In the meantime, all other ranks did not have the same test cases since their relative values were different.

The APFD calculation of the Bank program was computed as follows:

$n = 40$, $m = 28$

$APFD = 1 - \frac{76}{40 \times 28} + \frac{1}{2 \times 40} = 0.944642857$

For the Bank program, the value of APFD using both distance algorithms' fault matrix tables resulted in the same value of 0.94. The study obtained almost the same APFD value as all first ten test cases and the same priority rank, as shown in Table 6. On the other hand, for the other test case where the same Manhattan and Jaccard distance was applied during the prioritizing process, it could be observed that some test cases obtained different priority ranks.

Table 6
Priority List for Bank Program

Rank	Manhattan Distance priority list		Jaccard Distance priority list	
	Test case no.	Manhattan Distance value	Test case no.	Jaccard Distance value
1	TC 1	\	TC 1	\
2	TC 8	\	TC 8	\
3	TC 16	6	TC 16	0.16
4	TC 17	6	TC 24	0.16
5	TC 24	6	TC 17	0.8
6	TC 25	\	TC 25	\
7	TC 21	\	TC 21	\
8	TC 40	\	TC 40	\
9	TC 39	14	TC 39	1
10	TC 36	14	TC 36	1

7. Conclusion

This research conducted comparative experiments between two different string distance algorithms using two Java subject programs: Circular Queue and Bank. Higher APFD prioritizing test case value based on the Manhattan distance was obtained in the Circular Queue program, indicating that it is more effective. However, the experiment does suffer from limitations. The program samples

were comparatively small with a simpler structure, as they only consisted of one class. Moreover, the weight of the test cases was only computed using one of the six factors of the MFWA technique due to time limitations. Further work shall focus on applying their results from distance calculation in more complicated programs with different structures, implementing the Manhattan distance algorithm on Multi-Factor Weighted Approach (MFWA), and evaluating its performance. However, the result of this research shows that Manhattan Distance can obtain a higher APFD value than Jaccard Distance as proof that Manhattan Distance can detect faults earlier.

Acknowledgement
The authors humbly acknowledge the Encouragement Grant awarded by Universiti Teknologi Malaysia, Malaysia, with No Vot. Q.J130000.3851.19J69.

References

[1] Ahmad, Johanna. “Multi-factor Approach to Prioritise Event Sequence Test Cases.” (2018).
[2] Ahmad, Johanna, Salmi Baharom, Abdul Azim Abd Ghani, Hazura Zulzalil, and Jamilah Din. “Measuring the Efficiency of MFWA Technique for Prioritizing Event Sequences Test Cases.” *International Journal of Advanced Trends in Computer Science and Engineering*. (2019): 231–37.
[3] Askarunisa, MS A., MS L. Shanmugapriya, and DR N. Ramaraj. "Cost and coverage metrics for measuring the effectiveness of test case prioritization techniques." *INFOCOMP Journal of Computer Science* 9, no. 1 (2010): 43-52.
[4] Chen, Junjie, Yanwei Bai, Dan Hao, Yingfei Xiong, Hongyu Zhang, Lu Zhang, and Bing Xie. "Test case prioritization for compilers: A text-vector based approach." In *2016 IEEE international conference on software testing, verification and validation (ICST)*, pp. 266-277. IEEE, 2016. https://doi.org/10.1109/ICST.2016.19
[5] Elbaum, Sebastian, Alexey G. Malishevsky, and Gregg Rothermel. "Test case prioritization: A family of empirical studies." *IEEE transactions on software engineering* 28, no. 2 (2002): 159-182. https://doi.org/10.1109/32.988497
[6] Elbaum, Sebastian, Gregg Rothermel, and John Penix. "Techniques for improving regression testing in continuous integration development environments." In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 235-245. 2014. https://doi.org/10.1145/2635868.2635910
[7] Fang, Chunrong, Zhenyu Chen, Kun Wu, and Zhihong Zhao. "Similarity-based test case prioritization using ordered sequences of program entities." *Software Quality Journal* 22 (2014): 335-361. https://doi.org/10.1007/s11219-013-9224-0
[8] Abd Halim, Shahliza, Dayang Norhayati Abang Jawawi, and Muhammad Sahak. "Similarity distance measure and prioritization algorithm for test case prioritization in software product line testing." *Journal of Information and Communication Technology* 18, no. 1 (2019): 57-75. https://doi.org/10.32890/jict2019.18.1.4
[9] Hamilton, Thomas. "What is software testing? Definition, basics & types in Software Engineering." *Guru99. com* (2021).
[10] Jiang, Bo, and Wing Kwong Chan. "Input-based adaptive randomized test case prioritization: A local beam search approach." *Journal of Systems and Software* 105 (2015): 91-106. https://doi.org/10.1016/j.jss.2015.03.066
[11] Khan, Mohd Ehmer. "Different forms of software testing techniques for finding errors." *International Journal of Computer Science Issues (IJCSI)* 7, no. 3 (2010): 24.
[12] Astuti, Sinta Indi, Septo Pawelas Arso, and Putri Asmita Wigati. "Analisis standar pelayanan minimal pada instalasi rawat jalan di RSUD Kota Semarang." *Jurnal Kesehatan Masyarakat* 3, no. 1 (2015): 103-111.
[13] Anupriya, Ajeta. "Software Testing-Principles, Lifecycle, Limitations and Methods." *International Journal of Science and Research* 3, no. 10 (2014): 1000-1002.
[14] Rao, D. Nageswara, M. V. Srinath, and P. Hiranmani Bala. "Reliable code coverage technique in software testing." In *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*, pp. 157-163. IEEE, 2013. https://doi.org/10.1109/ICPRIME.2013.6496465
[15] Ledru, Yves, Alexandre Petrenko, Sergiy Boroday, and Nadine Mandran. "Prioritizing test cases with string distances." *Automated Software Engineering* 19 (2012): 65-95. https://doi.org/10.1007/s10515-011-0093-0
[16] Rao, D. Nageswara, M. V. Srinath, and P. Hiranmani Bala. "Reliable code coverage technique in software testing." In *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*, pp. 157-163. IEEE, 2013. https://doi.org/10.1109/ICPRIME.2013.6496465
[17] Sharma, Neha, and G. N. Purohit. "Test case prioritization techniques “an empirical study”." In *2014 International Conference on High Performance Computing and Applications (ICHPCA)*, pp. 1-6. IEEE, 2014. https://doi.org/10.1109/ICHPCA.2014.7045344

[18] Zhang, Jie. "Scalability studies on selective mutation testing." In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, pp. 851-854. IEEE, 2015. https://doi.org/10.1109/ICSE.2015.276

[19] Zhou, Jianyi, and Dan Hao. "Impact of static and dynamic coverage on test-case prioritization: An empirical study." In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 392-394. IEEE, 2017. https://doi.org/10.1109/ICSTW.2017.74

[20] Singh, Yogesh, Arvinder Kaur, Bharti Suri, and Shweta Singhal. "Systematic literature review on regression test prioritization techniques." *Informatica* 36, no. 4 (2012).

[21] Tanwani, L., & Waghire, A. "Test Case Prioritization for Regression Testing of GUI." *International Academy of Engineering and Medical Research* (1) (2016). http://www.iaemr.com/wp-content/uploads/2016/11/test-case-prioritization-regression-testing-gui.pdf

[22] Ulbert, Zs. "Software development processes and software quality assurance." *University of Pannonia.[online] Available at: http://moodle. autolab. uni-pannon. hu/Mecha_tananyag/szoftverfejlesztesi_folyamatok_angol/ch12. html [Accessed 30 April 2018]* (2014).

[23] Wohlin, Claes, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Vol. 236. Berlin: Springer, 2012. https://doi.org/10.1007/978-3-642-29044-2

[24] Sahak, Muhammad, S. A. Halim, Dayang Norhayati Abang Jawawi, and Mohd Adham Isa. "Evaluation of software product line test case prioritization technique." *International Journal on Advanced Science* 7, no. 4-2 (2017): 1601-1608. https://doi.org/10.18517/ijaseit.7.4-2.3403

[25] Hamlet, Richard G. "Testing programs with the aid of a compiler." *IEEE transactions on software engineering* 4 (1977): 279-290. https://doi.org/10.1109/TSE.1977.231145