Automation Architecture for Bayesian Network Based Test Case Prioritization and Execution

Ekincan UFUKTEPE
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkey
ekincanufuktepe@iyte.edu.tr

Tugkan TUGLULAR
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkey
tugkantuglular@iyte.edu.tr

***Abstract*—An automation architecture for Bayesian Network based test case prioritization is designed for software written in Java programming language following the approach proposed by Mirarab and Tahvildari [2]. The architecture is implemented as an integration of a series of tools and called Bayesian Network based test case prioritization and execution platform. The platform is triggered by a change in the source code, then it collects necessary information to be supplied to Bayesian Network and uses Bayesian Network evaluation results to run high priority unit tests.**

Keywords—testing; test prioritization; Bayesian Networks; Unit Testing

I. INTRODUCTION

Ouriques stated that testing approximately consumes 50% of the total projects budget [1]. Therefore, while performing tests, it is critical to decide which test cases should be performed in which order to maximize effectiveness of test process. State of the art in software development expects, with each commit request, unit tests should be performed automatically and if any fault is found, the developer should fix it before committing. While fixing the source code, modifications made may have some affect on other parts of the code. Intuitively, we can say that if modified part of the code has successfully passed from the unit tests then next concern will be to re-run the unit tests of the possibly affected part of the code. If an automated solution tells the tester in which order the unit tests should be re-run, this will save *time* and *effort*.

In this work, a Bayesian Network (BN) based test case prioritization approach proposed by Mirarab and Tahvildari [2] has been automated for Java programs and their unit tests. Additionally, effects of parameter selection (i.e. α, β, and δ) on test case prioritization has been observed and reported on a small case study with 4 classes and 204 lines of code. Table I shows our contributions to the proposed method in [2] in addition to making it fully automated including re-runing unit tests above the selected threshold.

The paper is organized as follows. Section II gives related work. BN based test case prioritization approach proposed by Mirarab and Tahvildari [2] is explained in Section III while comparing their tool usage with ours. Section IV presents our proposed automation architecture for BN based test case prioritization. Section V describes the case study and reports our observations on the effects of parameter selection. Section VI concludes the paper with future work.

TABLE I. Contributions

		Proposed work [2]	Our Investigations and Contributions
Used Tools	Collecting Software Metrics (CBO, Fan-Out)	ckjm	ckjm, ObjectAid UML
	Change Information	Sandmark	reJ
	Coverage Information	Emma	EclEmma
	Constructing Bayesian Network	Smile Library	OpenMarkov
	Unit Testing	- (none)	Junit
Methodology	Framework Implemented	Yes[2][3]	Yes
	Automatized Tool Implemented	No	Yes
	Observing Types of Source Code Changes Effect on BN Test Case Priority Results	No	Yes
	Observing the effectness of α, β and δ values on Test Case Priority Results	No	Yes
	End-to-end automation	No	Yes

II. RELATED WORK

Mirarab et al. [2] proposed an approach to prioritize test cases in aspect of regression testing to enhance the rate of fault detection. They have proposed a unified model based on probability that uses Bayesian Networks (BN). Their proposed model utilizes data of source code changes, software fault proneness and test coverage. The approach has been compared with nine different prioritization approaches by using APFD (Average Percentage Faults Detected) their metric results. Observations have shown that, BN has performed better results, when the software system contains more faults.

Wang et al. [4] claims that BN based Test Case Prioritization technique have focused on assessing the fault detection capability of each test case can utilize source code change information, software quality metrics and test coverage data. However, they claim that these techniques still has an absence that overlooks the similarities between test cases. To

mitigate this problem, they proposed a hybrid regression test case prioritization technique to achieve better prioritization by incorporating code coverage based clustering approach with BN test case prioritization to reduce the similar test cases having common code coverage. To find the similar test cases that have the common code coverages, code coverage based clustering approach has been used.

Xing et al. [5] have revealed a problem that most of current regression test case prioritization researches neglect to use internal structure information of software, although it is a significant factor influencing the prioritization of test cases. They have proposed an alternate regression test prioritization approach by considering the internal structure information and fault propagation behavior of modifications with respect to the modified versions of service-oriented workflow applications. Their approach schedules test cases based on dependency analysis of internal activities in service- oriented workflow applications.

Konsaard and Ramingwong [6] have used Genetic Algorithms (GA) to select and order test cases to provide a total coverage based on regression testing. Coverage is measured through the path that a test case passes and to obtain path information, a graph structure is needed. Therefore, Konsaard and Ramingwong [6] has used static analysis to get a CFG (Control Flow Graph) of the code. From the CFG a decision-to-decision graph created, thereby a test case's all possible paths could be easily generated. After a huge amount of test cases and paths are received, by using GA they optimize their test cases to create a total coverage. They have used and APCC (Average Percentage Code Coverage) metric to compare 5 other prioritization approaches. While the other 5 prioritization approaches APCC value change between 94.50% - 99.83% with 0.020 - 0.050 seconds of execution time, their approach has shown 100% APCC and 0.011 seconds of execution time.

Hao et al. [7] proposed an additional coverage-based technique, that uses a greedy strategy. They have investigated how much difference there are between the order produced by the additional technique and the optimal order in terms of coverage, they performed a their study on various empirical properties of optimal coverage-based test-case prioritization. To achieve the optimal order in acceptable time for their experimented programs, they formulated an optimal coverage-based test-case prioritization with integer linear programming (ILP) problem. Then they performed an empirical study comparing the optimal approach with the simple additional coverage-based approach. From this empirical study, they have observed that the optimal approach can only relatively outperform the additional coverage-based approach with no statistically significant difference in terms of coverage, and the latter significantly outperforms the former in terms of either fault detection or execution time. As the optimal approach schedules the execution order of test cases based on their structural coverage rather than detected faults, in addition they implemented the ideal optimal test-case prioritization approach, which schedules the execution order of test cases based on their detected faults. They took this ideal approach as the upper bound of test-case prioritization and conducted with another empirical study comparing the optimal approach and the simple additional approach with this alternate approach. Out of this empirical study, both the optimal approach and the additional approach significant outperform the alternate approach in terms of coverage, however, the latter significantly outperforms the former two approaches in terms of fault detection.

Hettiarachchi et al. [8] has described the use of system requirements and their risks that enables software testers to identify more important test cases that can reveal the faults associated with system components. Therefore, by using a fuzzy expert system, they aim to make the requirements risk estimation process more systematic and precise by reducing subjectivity. Furthermore, they have given empirical results that show that their proposed approach could improve the effectiveness of test case prioritization. They have used requirements complexity, modification status, security, and size of the software requirements as risk indicators and used a fuzzy expert system to estimate the requirements risks. Then they have used a semi-automated process to gather the required data for their approach and to make the risk estimation process less subjective. The results of their study indicated that the prioritized tests based on their new approach can detect faults early, and also their approach can be effective at finding more faults earlier in the high-risk system components compared to the control techniques.

III. Bayesian Network Based Test Case Prioritization

Mirarab and Tahvildari [2] in their BN based test case prioritization approach proposed to structure the Bayesian Network in three levels with three types of different nodes as follows:

- **Class nodes (*ce*):** Holds information of changed percentage, which has two states, *Changed* and *Unchanged*.
- **Class fault proneness nodes (*fe*):** Is the child node of ce, has a one-to-one relationship. Holds information of faultiness related to the changed or unchanged information. Has two states, *Faulty* and *Non-Faulty*.
- **Test Case Nodes (*te*):** Test case node can have more than one parent, due to the fact that a test case can address to multiple classes or target multiple faults. These nodes have two states, *Success* and *Failure*. These states values are given by the success of their coverage. Test Case Nodes are the indicator nodes for prioritizing them by their probability of success.

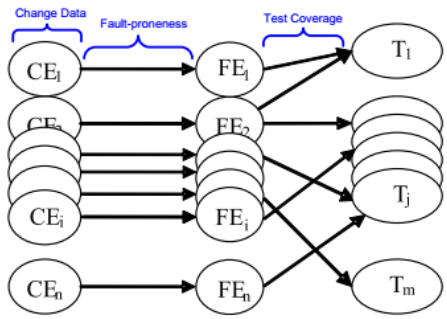

Fig. 1. Bayesian Network Structure [2].

To construct and execute the BN, it is important to collect the correct and consistent information to achieve viable results from the BN. Since there are three types of nodes in the BN, there are three types of information to be gathered from the software under consideration; *change information*, *coupling between objects (CBO) & fan-out information*, and *test case coverage information*.

A. Change Information

The change information is collected by change difference from the previous version of the project and the current project. Difference between these two projects is calculated from changed lines. However, the calculation of changed lines is not performed over SLOC (Source Lines Of Code) of two project, due to the unreliability of this metric can change by the coding style of the coder. Therefore, the calculation of changed lines is calculated over the changed lines of bytecode.

The calculation of changed lines of bytecode is given in Formula (1) [2].

$$Changed\ Percentage\ (ce) = \frac{\#\ of\ changed\ lines}{\#\ of\ total\ lines} \quad (1)$$

To collect the information of changed percentage, *Sandmark* is utilized in [2] whereas we use an open source tool called *reJ* [9].

B. Coupling between Objects & Fan-out Information

The coupling between objects is a metric that is defined by Chidamber and Kemerer [10] that count the number of classes that are coupled to. This metric is used as the indicator of fault-proneness. If a class has too many objects coupled with other classes, it is high probable that will be endangered by changes of other classes. To calculate the fault proneness under changed circumstances Formula (2) [2] has been used. Regardless the CBO value, the constant α sets the upper bound fault-introduction probability and constant δ_1 sets the minimum for the fault-introduction probability, when the class is changed.

$$P(fe_i = Faulty \mid ce_i = Changed) = \frac{\alpha CBO(e_i)}{\max(CBO(e_x))} + \delta_1\ , \quad (\alpha + \delta_1 \leq 1) \quad (2)$$

However, if there is no change in the classes, to calculate the fault proneness the fan-out metric has been used with the given Formula (3) [2]. The probability of fault proneness should be less than the fault proneness under changed circumstances. Therefore, the probability has been adjusted with the change impact factor. Similar to Formula (2) [2], but the case that the class has not changed (unchanged), β sets the upper bound and δ_2 sets the minimum bound of fault introduction probability.

$$P(fe_i = Faulty \mid ce_i = Unchanged) = \frac{\beta fan\text{-}out(e_i)}{\max(fan\text{-}out(e_x))} + \delta_2, \quad (\beta + \delta_2 \ll \alpha + \delta_1 \leq 1) \quad (3)$$

For CBO and fan-out calculation, *ckjm* is utilized in [2] whereas we use *ckjm* [11] and *ObjectAid UML* [12], respectively.

C. Test Case Coverage Information

To measure the success of test case and prioritize them, as a metric the coverage percentage of test cases has been used. The coverage calculations are based on classes instead of methods, due to the very small effect of difference they have shown. In addition test case coverage based on classes provides a reduction of nodes in BN that could affect the complexity and size of the node probabilistic tables (NPT).

A test case can be used in and cover other classes as well. Therefore, a test case might have several classes connected to itself. This introduces many combinations of classes with a 2^n size of NPT. It might be difficult to calculate and observe every case in the NPT. For these kinds of situations each node's (class's) coverage is observed independently and a noisy-OR is applied to calculate the other cases. The probabilistic formula for calculating test case success probability is given in Formula (4) [2].

$$P(t_i = Success \mid fe_j = Faulty) = Coverage(t_i, e_j) \quad (4)$$

To obtain the test case coverages, *Emma* is utilized in [2] whereas we use an *Eclipse* plug-in called *EclEmma* [13] with *JUnit*.

IV. Proposed Automation Architecture

In Figure 2, the architecture of our implemented automation tool for prioritizing test cases using BN is given. The architecture shows the process of how the test case prioritization tool works and which external support tools are used in which process.

First the user selects the previous and current version of projects. Then automatically obtains class CBOs, class relationships (fan-out), differences between classes, previous test case coverage information.

To collect the information of changed percentage, we use an open source tool called *reJ* [9]. reJ is a tool that allows visibility into Java classes for inspection and manipulation of Java class files. It provides the difference between two class files by comparing them. reJ tool only shows the difference

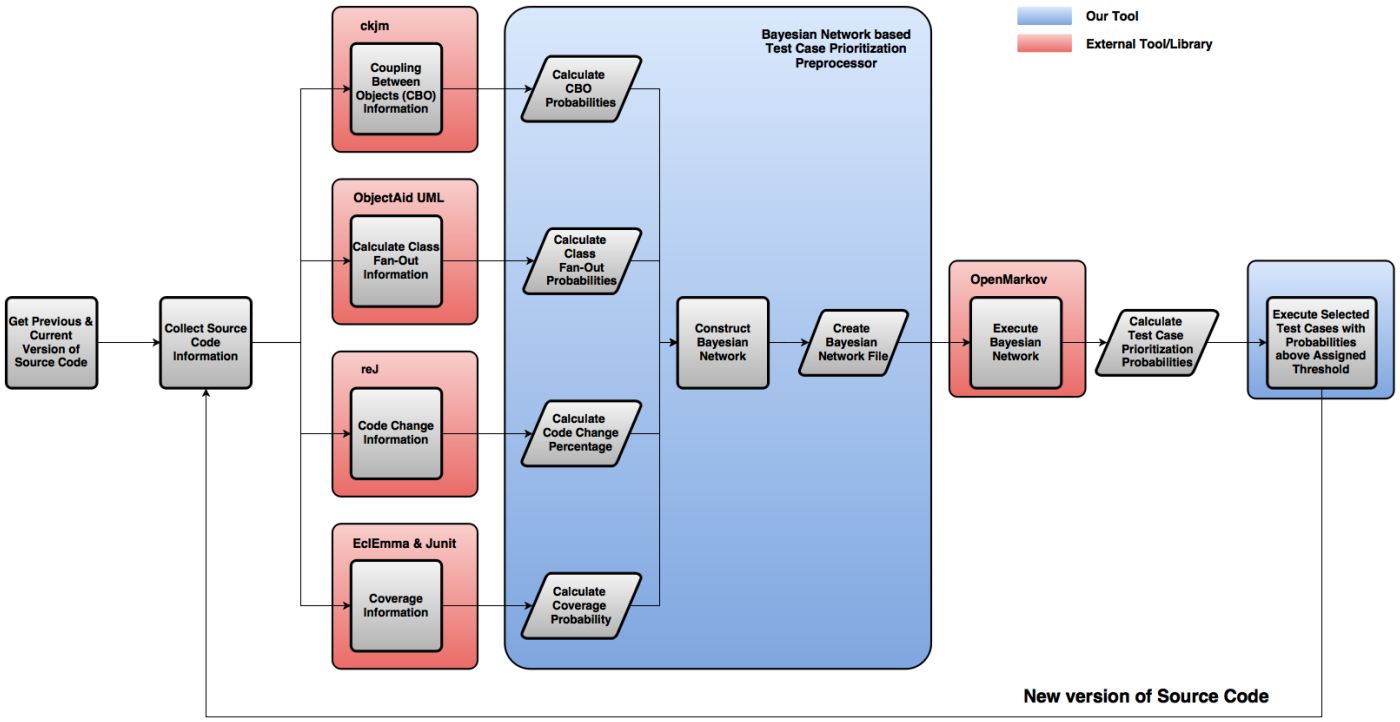

Fig. 2. Architecture of Bayesian Network based Test Case Prioritization Preprocessor

between classes, however it does not provide a measurement information of change percentage. Therefore, reJ has been modified that automatically calculates the change percentages between class files. The modification is implemented to assign the calculated change information directly to the BN. With our modifications reJ has integrated to our automated tool to calculate the change percentages that has been made in the class files.

For CBO and fan-out calculation, we use *ckjm* [11] and *ObjectAid UML* [12], respectively. Both ObjectAid UML and ckjm provides a XML output of the analyzed project. Therefore, it is easy to parse the output and adaptable for integrating into our tool to calculate the given Formulae (2) and (3).

To obtain the test case coverages, we use an *Eclipse* plug-in called *EclEmma* with *JUnit*. EclEmma provides a report of test case coverage result. After the unit testing is executed and coverages are calculated the results are reported with and .csv format file. The file is parsed with our developed tool and fed into the Bayesian Network.

By using the extracted information, node probabilities are calculated. To assign the calculated probabilities, the appropriate BN constructed. The BN construction creates the necessary nodes and establishes links between them, which satisfies the model. After BN structure is created, a BN XML file automatically created and written by our tool, which keeps the nodes, node relationships and probabilities, thereby the BN can be executed and return the results of test case priorities.

The BN is executed with an Java written open source tool called OpenMarkov [14]. After BN is executed with, OpenMarkov provides probabilistic results of test case selection. The test case that has the highest *Selected* probability, has the highest priority. When the test cases are prioritized the sorted test cases are executed by their prioritized order. On the new commitment of new version of the project, the process is repeated. The only manual part of the tool is selecting the current and previous version of the project.

V. Case Study

In this section, we discuss and investigate the type of changes that affects the test case priority results. As our case study we work on a small Java project that calculates the shipment cost by using a decision table. Information about the Java project and contents are given in Table I. The class diagram is illustrated in Figure 3.

TABLE I. Shipment Cost Java Project Information.

Class Name	SLOC	Number of Methods	Number of Test Cases
DecisionTable.java	73	1	12
Menu.java	64	4	4
Shipment.java	58	7	11
DeliveryDay.java	9	0	3

We define five different source code change scenarios for *Shipment Cost* Java Project as given below:

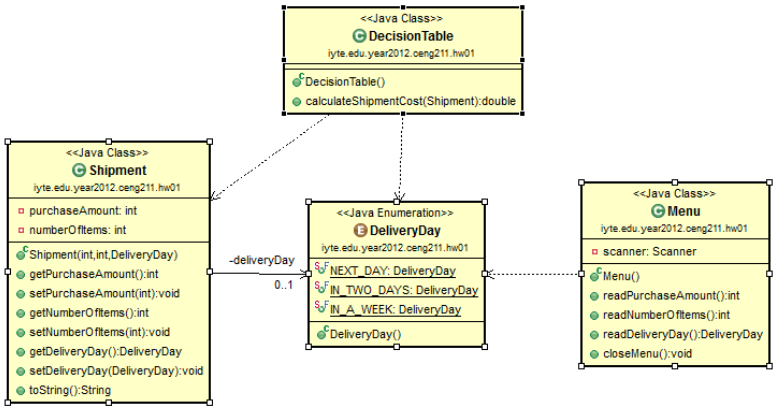

Fig. 3. Class Diagram of the Java Project used in Case Study

- **Scenario #1:** No change in any class.
- **Scenario #2:** A single class *DecisionTable.java* changed. A new parameter named *discount* is added and control for discount has been added in method. (0.69 change calculated)
- **Scenario #3:** A single class *DecisionTable.java* changed. A variable named *discount* is added and control for discount has been added in method. The *discount* variable has the same usage in Scenario #2 and has the same controls as well. (0.16 change calculated)
- **Scenario #4:** A single class *DecisionTable.java* changed. Only the cost variable's initial value has been changed from 0 to 25. (0.07 change calculated)
- **Scenario #5:** *DecisionTable.java*, *Shipment.java* and *Menu.java* have been changed and adapted to discount. (by order above 0.13, 0.30 and 0.22 changes calculated)

In Table II, for defined α=0.8, β=0.125, δ_1=0.1 and δ_2=0.1 values in [2]it has been observed that, there has been a peak in class *TestDecisionTable* priority result in scenario #2 compared with other scenarios. Scenario #2 and scenario #3 are actually representing code change that results in the same program behavior. However, scenario #2 prefers to get the *discount* input as a parameter, while scenario #3 prefers to get the *discount* input as a local variable from the user. By adding a new parameter in scenario #2, the method's actual structure and definition has been changed. Adding a new parameter to a method carries higher risks of affecting other places where it uses the method. It enforces the method's callers to be changed, otherwise by not applying or adapting the changes in callers will trigger faults.

As shown in Table III, different α parameter values defined in Formulae (2) and (3) has been tested in five scenarios, while δ_1, δ_2 and β values are set to fixed values as 0,05. These fixed values mean that, when there is no change in the class, the minimum and upper bound of fault-introduction probabilities are set to 0,05. On the other hand, when a class has been changed, we set the minimum bound of fault-introduction probability to 0,05 and let α, the upper bound of fault-introduction probability to be changed.

TABLE II. Calculated Test Case Prioritization Results.

Test Classes	Scenario #1	Scenario #2	Scenario #3	Scenario #4	Scenario #5
TestDecisionTable	0,333	0,6774	0,4153	0,369	0,4193
TestMenu	0,1943	0,1943	0,1943	0,1943	0,2547
TestShipment	0,209	0,209	0,209	0,209	0,3008
TestDeliveryDay	0,065	0,065	0,065	0,065	0,065

For each given α value in scenario #1, without any changes performed on the source code, no change in test case priority results has been observerved. Mirarab et al. [2] set the parameters values in Forumulae (2) and (3) as α*=0.8*, δ_1*=0.1*, δ_2*=0.1* and β*=0.125*. The difference between (α + δ_1) and (β +δ_2) is calculated *0.625*, while our differences are calculated by its order in Table III are; *0.85*, *0.65*, *0.45*, *0.25* and *0.05*. We see that where α*=0.7*, the difference between (α + δ_1) and (β +δ_2) is calculated as *0.65*, which is close to Mirarab's [2] difference result. Comparing with Mirarab's [2] results in scenario #1 (Table II.) with α*=0.7* (Table III.), the maximum fluctuation is observed in TestDecisionTable with 0.16, while the minumum fluctuation is observed is in TestDeliveryDay with 0.03. As a result we finalized that for α*=0.7* among all test case priority results, there is a fluctuation between $\pm$0.03-0.16 in test case priorities, compared with Mirarab's [2] α, β, δ_1 and δ_2. Therefore, satisfying the difference is not sufficient between (α + δ_1) and (β +δ_2) is not sufficient to obtain similar and close results. If β value was incremented, the affect on test case priorities for scenario #1, would increase the test case probabilities (Table III.). We conclude that defining the minimum bound of fault-introduction probability also has an impact on test case priority results, especially when there are minimal amount of change is made in the source code.

TABLE III. α Change Effect on Test Case Priorities

	Test Classes	α = 0,9	α = 0,7	α = 0,5	α = 0,3	α = 0,1
Scenario #1	Test DecisionTable	0,1704	0,1704	0,1704	0,1704	0,1704
	Test Menu	0,0925	0,0925	0,0925	0,0925	0,0925
	Test Shipment	0,099	0,099	0,099	0,099	0,099
	Test DeliveryDay	0,0325	0,0325	0,0325	0,0325	0,0325
Scenario #2	Test DecisionTable	0,6885	0,5666	0,4447	0,3228	0,2009
	Test Menu	0,0925	0,0925	0,0925	0,0925	0,0925
	Test Shipment	0,099	0,099	0,099	0,099	0,099
	Test DeliveryDay	0,0325	0,0325	0,0325	0,0325	0,0325
Scenario #3	Test DecisionTable	0,2905	0,2623	0,234	0,2057	0,1774
	Test Menu	0,0925	0,0925	0,0925	0,0925	0,0925
	Test Shipment	0,099	0,099	0,099	0,099	0,099
	Test DeliveryDay	0,0325	0,0325	0,0325	0,0325	0,0325
Scenario #4	Test DecisionTable	0,2229	0,2106	0,1982	0,1858	0,1735
	Test Menu	0,0925	0,0925	0,0925	0,0925	0,0925
	Test Shipment	0,099	0,099	0,099	0,099	0,099
	Test DeliveryDay	0,0325	0,0325	0,0325	0,0325	0,0325
Scenario #5	Test DecisionTable	0,3489	0,2789	0,2473	0,2143	0,1799
	Test Menu	0,2526	0,1546	0,1358	0,1169	0,0981
	Test Shipment	0,3412	0,1931	0,1646	0,136	0,1076
	Test DeliveryDay	0,0325	0,0325	0,0325	0,0325	0,0325

VI. CONCLUSION

Contributions made to the approach presented in [2] are given in Table I. Excluding *ckjm*, different supporting tools has been used in our architecture and platform.

We haven't evaluated the prioritization results by using APFD metric and the compared with other prioritization techniques since it has already been evaluated and compared in [2]. For both an framework has been implemented, however, the in our work an automated tool has been implemented to ease the usability and collection of project data.

Bayesian Networks has shown promising results, compared with other prioritization techniques. Therefore, we put under scope Mirarab et al. [2]'s work. In this work, an automation platform has been implemented that uses a test case prioritization approach [2] that automatically obtains data from projects, create the BN and calculates test case priority probability results.

It has been observed that source code changes affect test case prioritization results. It is seen that changes perfomed over method's definition has a high impact on test case priority results, no matter if the content of the method is same with the previous method.

It has been observed that in Formula (2) and (3) the minimum and upper bounds of fault-introduction probabilities α, β, and δ values have an effect on test case prioritization results. While, β and δ values are kept same, α has been changed and new test case priority results are obtained. It has been investigated that satisfying the similar differences of minimum and upper bounds of fault introduction probability does not provide the same test case priority results. Therefore, regardless the difference between minimum and upper bound fault introduction probability values, the fault introduction probabilities by their own has a direct influence on test case priority results in BN.

REFERENCES

[1] J. F. S. Ouriques, "Strategies for Prioritizing Test Cases Generated Through Model-Based Testing Approaches," in *Proceedings - International Conference on Software Engineering*, 2015, vol. 2, pp. 879–882.

[2] S. Mirarab and L. Tahvildari, "A Prioritization Approach for Software Test Cases Based on Bayesian Networks," *Fundam. Approaches to Softw. Eng. Springer Berlin Heidelb.*, vol. 4422, pp. 276–290, 2007.

[3] S. Mirarab, "A Bayesian Framework for Software Regression Testing," 2008.(MSc. Thesis)

[4] X. Zhao, Z. Wang, X. Fan, and Z. Wang, "A Clustering – Bayesian Network Based Approach for Test Case Prioritization," *Comput. Softw. Appl. Conf. (COMPSAC), IEEE 39th Annu.*, vol. 3, pp. 542–547, 2015.

[5] H. Wang, J. Xing, Q. Yang, D. Han, and X. Zhang, "Modification Impact Analysis Based Test Case Prioritization for Regression Testing of Service-Oriented Workflow Applications," in *2015 IEEE 39th Annual Computer Software and Applications Conference*, 2015, pp. 288–297.

[6] P. Konsaard and L. Ramingwong, "Total Coverage Based Regression Test Case Prioritization using Genetic Algorithm," in *12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2015, pp. 1–6.

[7] D. Hao, L. Zhang, L. Zang, Y. Wang, X. Wu, and T. Xie, "To Be Optimal Or Not in Test-Case Prioritization," *IEEE Trans. Softw. Eng.*, vol. 6, no. 1, pp. 1–20, 2015.

[8] C. Hettiarachchi, H. Do, and B. Choi, "Risk-based test case prioritization using a fuzzy expert system," *Inf. Softw. Technol.*, vol. 69, pp. 1–15, 2016.

[9] "reJ": http://rejava.sourceforge.net/.

[10] S. R. Chidamber and C. F. Kemerer, "Towards a metric suite for object oriented design," in *Proceedings of the Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA)*, 1991, vol. 26, no. 11, pp. 197–211.

[11] D. Spinellis, "Tool writing: A forgotten art?," *IEEE Softw.*, vol. 22, no. 4, pp. 9–11, 2005.

[12] "ObjectAid UML" : http://www.objectaid.com.

[13] "EclEmma" : http://eclemma.org/.

[14] M. Arias, F. J. Diez, M. A. Palacios-Alonso, M. Yebra, and J. Fernández, "POMDPs in OpenMarkov and ProbModelXML," *Seventh Annu. Work. Multiagent Seq. Decis. Under Uncertain.*, no. June, pp. 1–8, 2012.