AUTOMATED TECHNIQUES FOR PRIORITIZATION OF METAMORPHIC RELATIONS FOR EFFECTIVE METAMORPHIC TESTING

by

Madhusudan Srinivasan

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

July 2022

ACKNOWLEDGEMENTS

I would like to thank my PhD advisor Dr. Upulee Kanewala for the kind support and guidance throughout my PhD Program. She has been very approachable all through my graduate years for research discussions and has made sure I stay on the right path in my research.

I would also like to thank Dr. David Millman for the very useful feedback and suggestions on my dissertation, Dr. Indika Kahanda for his guidance on machine learning, Dr. Sean Yaw, Dr. Clem Izurieta, and Dr. John Paxton for their service on my thesis committee. I also extend my gratitude to my lab mate Prashanta for participating in several discussions regarding my research in the last several years. I also thank my friends Venkat and Toral for their constant support and encouragement. Also, I would like to thank my family for their push and support.

My research is funded by the award number 1656877 from the National Science Foundation.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

TABLE OF CONTENTS – CONTINUED

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

LIST OF TABLES – CONTINUED

Table Page

LIST OF TABLES – CONTINUED

Table Page

LIST OF TABLES – CONTINUED

Table Page

LIST OF TABLES – CONTINUED

Table Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

ABSTRACT

An oracle is a mechanism to decide whether the outputs of the program for the executed test cases are correct. In many situations, the oracle is not available or too difficult to implement. Metamorphic testing is a testing approach that uses metamorphic relations (MRs), properties of the software under test represented in the form of relations among inputs and outputs of multiple executions, to help verify the correctness of a program. Typically, MRs vary in their ability to detect faults in the program under test, and some MRs tend to detect the same set of faults. In this work, we aim to prioritize MRs to improve the efficiency and effectiveness of MT. We present five MR prioritization approaches: (1) Fault-based, (2) Coverage-based, (3) Statement Centrality-based, (4) Variable-based, and (5) Data Diversity-based. To evaluate these MR prioritization approaches, we conducted experiments on complex open-source software systems and machine learning programs. Our results suggest that the proposed MR prioritization approaches outperform the current practice of executing the source and follow-up test cases of the MRs randomly. Further, our results show that Statement Centrality-based and Variable-based approaches outperform Code Coverage and random-based approaches. Also, the proposed approaches show 21% higher rate of fault detection over random-based prioritization. For machine learning programs, the proposed Data Diversity-based MR prioritization approach increases the fault detection effectiveness by up to 40% when compared to the Code Coverage-based approach and reduces the time taken to detect a fault by 29% when compared to random execution of MRs. Further, all the proposed approaches lead to reducing the number of MRs that needs to be executed. Overall, our work would result in saving time and cost during the metamorphic testing process.

CHAPTER ONE

INTRODUCTION

A test oracle is an essential component of testing that is used to distinguish between correct and incorrect behavior of a system under test (SUT). However, for complex systems such as scientific software or machine learning applications developing test oracles is practically difficult [43] and leads to a problem known as the *oracle problem* in software testing [9, 37, 48]. Metamorphic testing (MT) is an effective technique to alleviate the test oracle problem. Generally, testing involves verifying the output produced by a test case using a *test oracle*. MT involves verifying that the outputs of multiple executions of the SUT and their corresponding inputs satisfy a given *metamorphic relation (MR)* [19, 73]. Formally, an MR is a necessary property of the SUT and is specified over multiple inputs and their corresponding outputs [19]. The steps for conducting MT are as follows:

1. Identify MRs for the SUT using its specification or the programmer's knowledge about the SUT.

2. Create *source test cases* and execute them on the SUT. Existing test cases can be used as source test cases or using test generation approaches such as random generation.

3. Construct the *follow-up test cases* based on the MRs and execute them on the SUT.

4. If the source and follow-up test inputs and outputs do not satisfy a given MR, then the SUT is faulty.

Metamorphic testing has been successful in finding bugs in systems across various domains [17, 20, 38, 80]. Segura et al. [72] provided a survey on metamorphic testing and it was shown that metamorphic testing could be applied in multiple domains and found to have been effective in detecting many unknown faults in various systems. The survey provided an interesting case study, where MT could reveal 147 confirmed bugs in two open source C compilers, GCC and LLVM using the "equivalence preservation"metamorphic relation to test compilers [79]. Segura et al. [73] reported that MT approach detected a bug in Google Map, where the starting and ending point was two meters apart from each other but Google Map returned a route of 4.3 miles.

More recently, MT was used to reveal hidden defects in two widely used citation database systems, Scopus and Web of Science particularly when handling hyphenated paper titles [93]. Further, Zhou et al [94] proposed a symmetry metamorphic relation pattern to derive multiple concrete MRs that lead to revealing previously unknown failures in widely used applications. Additionally, MT was applied to conduct a large scale empirical study using four major web search engines: Google, Bing, Chinese Bing, and Baidu [95]. The experimental results of this study revealed that MT could reveal various types of failures that impact the quality of the results generated by a search [95]. Similarly, case studies have been conducted to select good MRs which are effective at finding the fault for the given SUT [18, 78]. The experiment result showed that the knowledge of the SUT is not adequate for distinguishing good MRs, rather good MRs make multiple executions of the program as different as possible. Therefore, with these successful applications of MT in different domains, it becomes important to develop techniques that allow using MT effectively for regression testing.

Prioritization of metamorphic relations is an important step during regression testing. Regression testing is conducted every time a program is modified to make sure that no new bugs were introduced during the modification process [71, 86, 88]. As agile software development methods are widely adapted, regression testing has to be done more frequently with releases of successive versions [66, 70]. Individual MRs significantly differ in their fault detection effectiveness and some MRs tend to detect the same type of faults [77]. Figure 1.1 provides a motivating example to show varying fault finding effectiveness across MRs for BBMap program. We can observe that the MRs one to eight have different fault finding effectiveness from lowest 3% to highest 60%. Checking a MR requires multiple executions of the SUT using the corresponding source and follow-up test cases. Also, machine learning programs exhibit non-deterministic behavior and suffer from test oracle problem [13, 91]. Further, using large dataset could take several days to train and generate a machine learning model. As a result, applying MT on scientific software's and machine learning systems involves extra execution overhead when compared to typical testing approaches, where the passing/failing is determined by a single execution of the SUT. Therefore, it becomes important to prioritize the MRs.

Hence we propose a set of approaches for MR prioritization: Fault-based, Code Coverage-based, Statement Centrality-based, Variable-based and Data Diversity-based approaches.

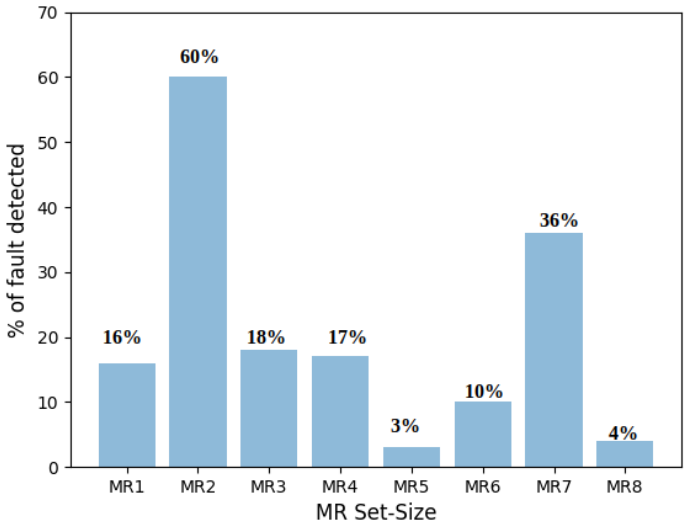


Figure 1.1: Fault detection effectiveness of individual MRs for BBMap

We make the following contributions:

1. Propose novel automated methods for *MR prioritization*. The methods are:

   - *Fault-based*, where fault detection effectiveness of MRs are used to prioritize them is presented in Chapter 3.
   - *Coverage-based*, where statement and branch coverage information of the MRs are used to prioritize them is presented in Chapter 3.
   - *Statement Centrality-based*, where we propose metrics to weigh statements in the PUT based on factors: statement affected, impacted, and statement infection propagation ability is presented in Chapter 4. The approach tries to remove some shortcomings of Fault-based and coverage-based MR prioritization.
   - *Variable-based method*, where we propose metrics based on internal and output variable values to prioritize MR is presented in Chapter 4. The

approach tries to remove some shortcomings of the statement-centrality-based approach.

- *Data Diversity-based method*, where we propose metrics to prioritize MRs based on the diversity in the dataset. These metrics were designed to prioritize MRs for machine learning-based applications and remove shortcomings of Statement Centrality and Variable-based approach.

2. Evaluate the effectiveness of our proposed MR prioritization methods using a complex real-world system and machine learning programs. .

3. Create prioritized lists of MRs that can be used for testing similar systems in diverse domains such as sequence alignment, supervised learning, and bio-entity recognition.

Therefore our MR prioritization approaches can be used to improve the effectiveness of MT over the current practice of random execution of MRs. The key benefits of our proposed approaches for MR prioritization are:

- Our proposed MR prioritization approaches lead to reducing the number of MRs needed to detect faults.

- Providing the ability to improve the rate of fault detection- a measure of how quickly a fault is detected during the testing process.

- MR quality score is provided for each of the MR and would help in improving the quality of the MR for future execution.

The remainder of the dissertation is organized as follows: In chapter 2, related works on the prioritization of MRs and background are presented. In Chapter 3, Fault-based and Code Coverage-based approaches for prioritization of MR in the

context of regression testing are presented. In Chapter 4, Statement Centrality and Variable-based approaches to prioritize MR are presented. In Chapter 5, a Data Diversity-based approach for prioritizing MRs for machine learning programs is presented. In Chapter 6, conclusion and future work is presented.

CHAPTER TWO

BACKGROUND & RELATED WORK

2.1 Background

In this Chapter, we provide some background on testing, metamorphic testing, program slicing, mutation testing and machine learning.

2.1.1 Software Fault

According to the IEEE Standard IEEE Std 7-4.3.2-2003 [7], a software fault is an incorrect step, process or data definition in a computer program. Software faults are static and caused by structural imperfection in a software system.

2.1.2 Testing

Testing is defined as the process of evaluating the software by observing the execution [5, 59, 85]. The test inputs do not always trigger the fault into incorrect output (test failure). Therefore, for a test case to observe a failure, three conditions must be satisfied:

- Reachability: The location in the program containing the fault must be reached.
- Infection: The program state for execution is the value of all the variables in the program. The fault is detected when the execution reaches the faulty location or the statement, that causes an incorrect program state, known as *infection* [22, 65].
- Propagation: The infected program state must propagate to the program output to be incorrect [82].

Figure 2.1 explains the reachability, infection and propagation (RIP) model. Statement centrality based method is based on the RIP model.

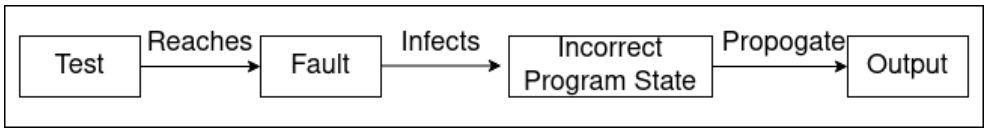


Figure 2.1: Reachability, Infection and Propagation Model

2.1.3 Metamorphic Testing

Metamorphic testing is used to alleviate the oracle problem. The main intuition behind metamorphic testing is that even in the absence of expected output for a input, it may be still possible to use relation among the outputs of multiple inputs to detect the presence of a fault in the program. The relations are known as metamorphic relations [21]. Metamorphic relations are used to derive a follow-up test case from the source test case and then later used to check whether the outputs satisfy the MR. An example of metamorphic testing is provided here.

Suppose an algorithm in a program P identifies the short distance between two vertices a and b in an undirected graph G. Let a be the source vertex and b be the destination vertex in the graph. The necessary property based on the algorithm could be: "If the source vertex a and destination vertex b is interchanged, the length of the shortest path will remain unchanged". The metamorphic relation based on the property could be: if the vertices a and b are interchanged, the length of the shortest path will remain unchanged, that is, $\mid f(G,b,a) \mid = \mid f(G,a,b) \mid$. Based on this metamorphic relation, source test case would be (G, a, b) and the follow-up test case would be (G, b, a). We now verify the result of multiple executions against the metamorphic relation, that is, $\mid P(G,b,a) \mid = \mid P(G,a,b) \mid$. The violation of metamorphic relation indicates the presence of a fault in P.

2.1.4 Program Slicing

Program Slicing is a technique in software testing which takes a set of program statements, *program slice*, which may affect a value at a particular point of interest [11, 84]. The slicing is of two types: static slicing and dynamic slicing. Static slicing contains all the statements that may affect the value of the variable at a particular point for any execution of the program [81]. Static slice considers every possible execution of the program. Dynamic slice contains all the statement that may affect the value of variable for a particular execution of the program [1, 45]. Our proposed approaches such as statement centrality and variable based approach uses dynamic slicing technique. Slicing can move into two directions: backward and forward. Backward and forward slice is computed from the program dependency graph by following the data and control dependencies.

2.1.4.1 Backward Slice

The backward slice identifies statements that affect the computation of a particular statement [33]. An example of a backward slice is provided in Figure 2.2. The backward slice from line 9 is lines 7, 5, and 4. Line 7 and 5 are included in the slice due to data dependency to line 9. Line 5 is included in the slice because data-dependent line 7 is control dependent on line 5.

2.1.4.2 Forward Slice

Forward slice identifies those statements that are affected by a particular statement [27]. Figure 2.3 provides an example of a forward slice. The variable sum defined in line 3 is used in line 6 and line 8. Also, line 6 is control dependent on line 5. Therefore, forward slice from line 3 is line 5, 6 and 8.

2.1.5 Mutation Testing

The effectiveness of the metamorphic testing approach is determined using mutation testing. Mutation testing involves modifying a program in small ways and

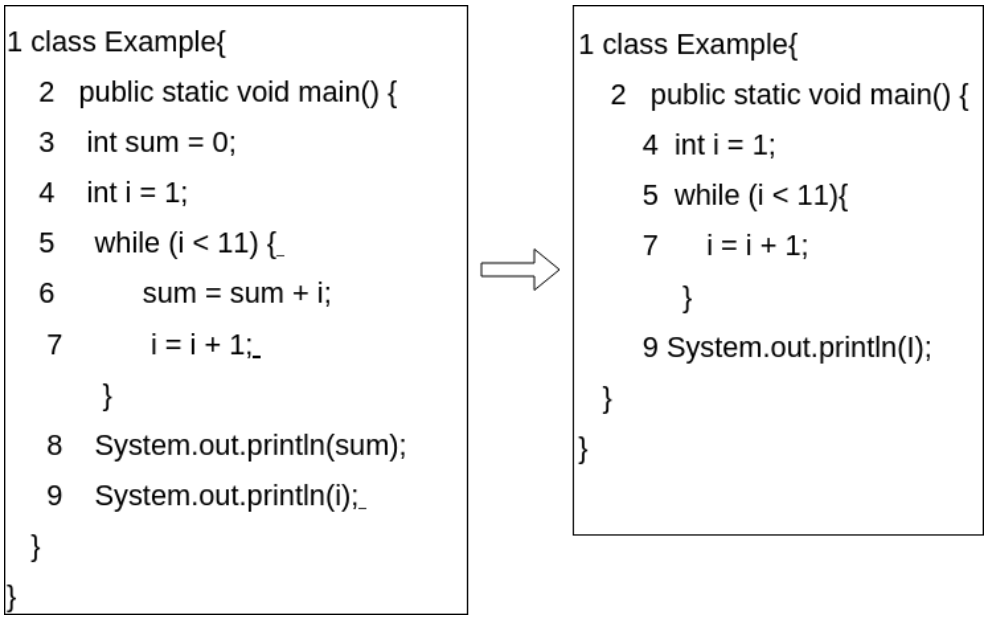


Figure 2.2: Left side of the figure is the original program and right side is the backward slice of the original program with respect to line number 9 and variable I.

each mutated version is called a mutant [23, 39, 53]. The quality of the test suite is decided by the percentage of mutants killed and alive. The mutant is considered killed if the mutant program output is different from the original program output and considered alive otherwise. The effectiveness of the test suite is determined based on the mutation score. The *mutation score* is calculated using the formulae.

$$\text{Mutation Score} = \frac{\text{Number of killed Mutants}}{\text{Total Number of Mutants}}$$

2.1.6 Machine Learning

Machine learning is an application of AI that enables computer programs to learn for themselves and improve from past experience and historical data without being explicitly programmed [58,83,90]. The data used by the machine learning algorithm is divided into two subsets: a *training set* and a *test set*. The training data set is used to create the predictive model and learn patterns, while the test data set is unseen data

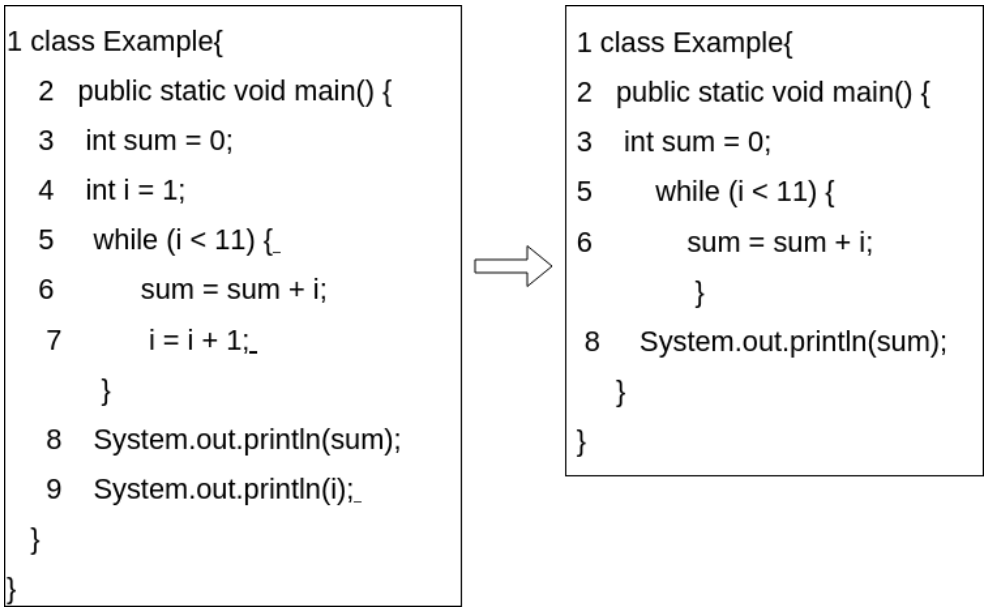


Figure 2.3: Left side of the figure is the original program and right side is the forward slice of the original program with respect to line number 3 and variable sum.

used to evaluate the performance of the predictive model. Within the field of machine learning, there are two main types of tasks: *supervised* and *unsupervised learning*. Supervised learning use labeled data set to train ML algorithms to predict and classify outcomes. The labeled dataset has output tagged to the corresponding input for the machine to predict unseen data. Supervised learning is used for classification and regression tasks. Unsupervised learning does not contain labeled datasets and finds a hidden pattern in unlabeled datasets and gives the required output [36,67,83]. Unsupervised learning is used for clustering task [8,34,74]. In our proposed approaches for prioritizing metamorphic relations, we investigated four ML algorithms, k-Nearest Neighbor (KNN), Linear Regression, Naive Bayes, and Convolutional Neural Network. The ML algorithms are explained in detail below.

2.1.6.1 K- Nearest Neighbors Algorithm: The k-nearest neighbors (KNN) algorithm is a supervised machine learning algorithm used to solve classification and

regression problems. A classification problem has a discrete value as its output. A regression problem has a real number as its output [92].

Let us consider data points plotted from our training set as shown in Figure 2.4. The training set contains a total of 3 red and 3 blue data points. Red data points are classified as class A and blue data points are classified as class B. The yellow data point is the new point for which a class is to be predicted. To predict the class for the new data point, we apply K nearest neighbor algorithm. The nearest neighbors are those data points that have a minimum distance from our new data point. K is the count of the nearest neighbor to include in the majority of the voting process. Distance is calculated by using the Euclidean, Hamming distance, Manhattan distance, or Minkowski distance. The class for the new data point is predicted based on the class label assigned to the majority of the K nearest neighbors in the training dataset.

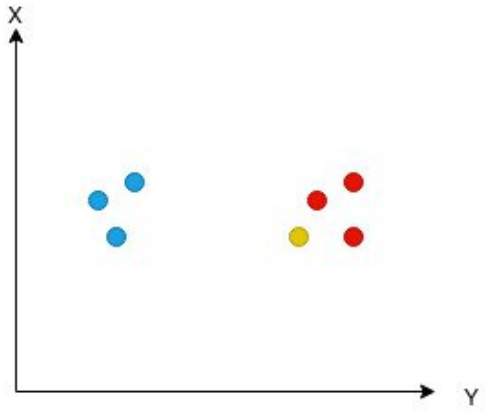


Figure 2.4: KNN Data Points

2.1.6.2 Linear Regression: Linear regression is a linear model, that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x). Based on the given data points, a line is plotted that models the points

the best [12, 15]. The line can be modeled based on the linear equation shown below.

$$y = a_0 + a_1 * x \tag{2.1}$$

where where a0 is the intercept, a1 is the coefficient or slope of the independent variable x and y is the dependent variable.

2.1.6.3 Naive Bayes: Naïve Bayes is a probabilistic machine learning algorithm based on the Bayes Theorem, used in a wide variety of classification tasks. Bayes Theorem is used for calculating conditional probabilities. Conditional probability is a measure of the probability of an event occurring given that another event has occurred [10, 40]. The formula is below:

$$P(A \mid B) = \frac{P(B|A).P(A)}{P(B)} \tag{2.2}$$

Where $P(B \mid A)$ represents the probability of B occurring given evidence A has occurred, $P(A \mid B)$ represents the probability of A occurring given the evidence B has occurred and P(A) represents the probability of A occurring.

2.1.6.4 Convolutional Neural Network: A convolutional neural network is a feed-forward neural network that is generally used to analyze visual images by processing data and often with up to 20 or 30 layers. It's also known as a ConvNet. A convolutional neural network is used to detect and classify objects in an image. The power of a convolutional neural network uses a layer called the convolutional layer [4, 44, 64]. Figure 2.5 provides an example of a convolutional neural network. As shown in the figure, C1 is the first convolutional layer. This layer takes the image

as the input and C1 outputs six images of size 28x28. S2 is known as the average pooling layer and it scales down the six 28x28 images by producing six output images of size 14x14. C3 is the second convolutional layer which takes the six 14x14 images and produces 16 images of size 10x10. S4 is the second average pooling layer which scales down the sixteen 10x10 images to sixteen 5x5 images. C5 is a fully connected convolutional layer with 120 outputs. The output is a 1D array of length 120. F6 is a fully connected layer mapping the 120-array to a new array of length 10. Each element of the array corresponds to a handwritten digit 0-9. The output layer is a softmax function that transforms the output of F6 into a probability distribution of 10 values which sum to 1.

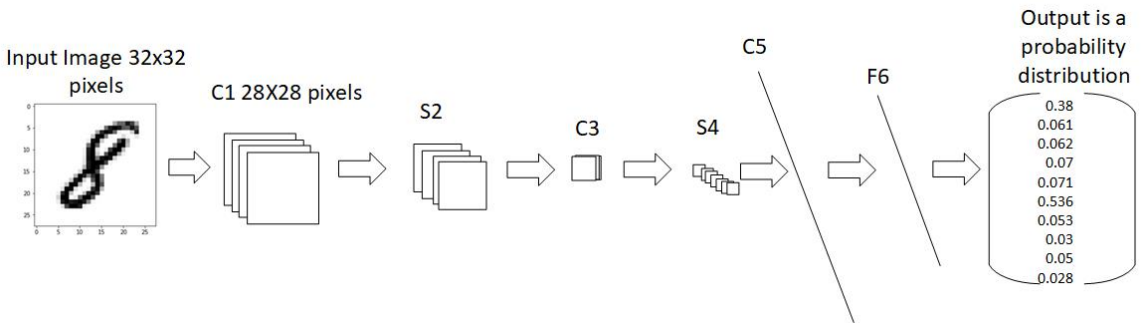


Figure 2.5: CNN Example

2.2 Related Works

Gay et al. [28] proposed a mutation-based approach for selecting a set of internal and external variables to be monitored during testing. In this approach, the variables are ranked based on the mutation killing rate. However, this approach would require the tester to specify the expected values for the variables selected as the oracles. Specifying the expected value for variables may not be possible for programs that face the oracle problem. In our prioritization approach, we use mutation technique and

code coverage strategies such as statement and branch coverage to prioritize MRs. Yu et al. [89] proposed regression test selection technique that focuses on specific classes of faults that can be detected by internal oracles during system execution. The approach uses program slicing to identify code changes that are appropriate to internal oracles, and selects test cases that cover these changes. The the results of an empirical study show that the proposed technique is more effective than other regression test selection techniques, relative to the classes of faults targeted by the internal oracles. In comparison, our proposed approaches focus on prioritizing metamorphic relations for regression testing based on internal variables since internal oracle is not available for many scientific software's and machine learning programs.

Loyola et al. [50] proposed an approach that supports the generation of test oracles. The approach ranks program variables based on the interactions and dependencies observed between them during program execution. Using this ranking, testers can construct assertion-based oracle. The approach first involves manually selecting variables which requires the support of developers and domain expert to understand the code base and then select required variables. In comparison, our approach automatically selects variables and applies it in the context of prioritizing metamorphic relations. Chen et al. [16] proposed static approach to oracle data selection. The approach constructs a probabilistic substitution graph based on the definition-use chains of the program under test. The graph helps to estimates the fault-observing capability of each candidate oracle data or variables, and finally selects a subset of oracle data with strong fault-observing capability. The experimental study on 11 subject systems written in Java and results demonstrates that static approach is more effective than state-of-the-art dynamic approaches. In comparison, our proposed variable based approach focuses on prioritizing metamorphic relations based on diversity in the internal variables values between source and follow-up test

cases of the MR.

Chen et al. [18] proposed guidelines for the selection of good MRs. In order to develop guidelines for MR selection, a case study was conducted on the shortest path program that implements the Dijkstra's algorithm. The experiment results suggest that the knowledge of the domain is not adequate for distinguishing good MRs, rather good MRs make multiple executions of the SUT that are different from each other. We address this issue by providing qualitative analysis of test cases execution profiles both in the presence and absence of code coverage. Asrafi et al. [6] conducted an empirical study to investigate the relationship between the execution behavior of the MRs and effectiveness of MRs. The study was based on the rationale that the higher the code coverage the more diverse the execution behaviors of the test set. The MRs were applied on two programs and it was found that MRs with low coverage have low effectiveness in detecting software faults. This work does not focus on prioritization of MRs but determines the effectiveness of MRs using code coverage. Spieker et al. [75] proposed a method to select MRs using reinforcement learning based on contextual bandit. The approach defines a test which sequentially selects a MR that is expected to provide the highest payoff, i.e., that is most likely to reveal faults. The MRs which are likely to reveal faults is learned from successive exploration trials. The experimental result on two image analysis suggest that the approach was effective for selecting MRs and is more time-efficient than exhaustive testing and more effective than the standard approach of pure random selection. We apply our proposed approach in the context of regression testing for prioritizing MRs.

Cao et al. [14] proposed six metrics to measure the dissimilarity between source and follow-up test cases. They conducted an empirical study to measure the dissimilarity between source and follow-up test case executions using these measures. The work does not employ the prioritization of MRs for regression testing and

determines dissimilarity quantitatively. However, each statement in the program shows a different level of importance to the program output during execution. For example, statements having variable definitions without a corresponding use or an assert statement do not contribute to the program output during execution and hence the importance of those statements is minimum. The proposed metrics could be an issue in the absence of code coverage or test cases in metamorphic relations show identical code coverage.

Mayer and Guderlei [57] conducted an empirical study to evaluate the usefulness of MRs. Based on the experiment result, the authors devise several criteria to asses the quality of the MRs based on the potential usefulness. These criteria were defined based on the different execution paths undertaken by the source and follow-up test cases. While this work provides some directions into selecting useful MRs, they do not directly address MR prioritization. Sun et al. [3] proposed a technique to generate good source test cases in metamorphic testing based on constraint solvers and techniques of symbolic execution. Also, a prioritization of source test cases was conducted based on path distances among test cases. This work does not focus on prioritization of metamorphic relations. In particular, we focus on developing automated methods for MR prioritization. Sun et al. [78] proposed feedback driven metamorphic testing based on historical execution information of source test cases to dynamically select metamorphic relations and source test cases with higher fault detection potential. The proposed approach was implemented as a comprehensive framework which consist of the basic MR components, historical testing data and testing strategies. The approach is designed for source test case selection and does not focus on prioritization of MRs for scientific softwares or machine learning programs.

Hui et al. [35] provided qualitative guidelines for MR prioritization. The guidelines could be used to select effective MRs and improve the performance of

MT. The author also provides three measurable metrics for MR selection: in-degree of MR, algebra complexity of MR, and distance between test inputs of MR. The empirical result shows inconsistency between the correlation value and the metrics, indicating that other factors could influence the effectiveness of MR.

Mani et al. [55] proposed metrics to identify the quality of dataset based on centroid positioning, pairwise boundary conditioning and equivalence partitioning. This work does not provide insight on the diversity in the dataset and does not address MR prioritization. Our work focuses specifically on prioritization of metamorphic relations. Nakajima et al. [62] proposed an approach to derive a set of metamorphic properties for machine learning program and presented a new test coverage called dataset coverage. The idea behind dataset coverage is to divide the input dataset into equivalence classes and pick up a representative data point from each equivalence class and form a test input. The approach does not focus on prioritization of MRs. Nakajima et al. [61] proposed a new metamorphic testing method applicable to neural network learning models based on dataset diversity and behavioral oracles. Dataset diversity method takes into account the dataset dependency of training results, and provides a way for generating follow-up test inputs. Behavioral oracles monitors changes of certain statistical indicators during the training processes and provides a basis of metamorphic relations to be checked. The approach was applied on classifying handwritten numbers and able to detect anomalies. In comparison, our approach works on applying data diversity based on various metrics to prioritization of MRs for ML programs.

CHAPTER THREE

FAULT AND COVERAGE-BASED APPROACH

In this Chapter, we cover two MR prioritization approaches and the Chapter is organized as follows: Section 3.1 provides the motivation for this work. In Section 3.2 and 3.3, the proposed approaches for MR prioritization is discussed in detail. In Section 3.4, the validation procedure for the empirical study is discussed. In Section 3.5, results and answers to the research questions are presented. In Section 3.6, potential threats to validity are discussed. In Section 3.7, a discussion of conclusions is presented.

3.1 Motivation

With the successful applications of MT in different domains, it becomes important to develop techniques that allow using MT effectively for *regression testing*. Regression testing ensures that the software works as expected after code changes, updates, or improvements to the existing software [5]. With the wide adaptation of agile software development methods, regression testing has to be done more frequently with releases of successive versions. Therefore, in such a development environment, it becomes beneficial to use efficient and effective *MR prioritization* approaches. In addition, previous studies have shown that typically five to nine MRs are used for testing [72] and the cost (in terms of time and resources) of the MT process is proportional to the number of MRs used for testing. SUT can have MRs with multiple source and follow-up test cases. Thus, as the number of MRs increases, the number of test cases can increase exponentially and as a result, the time taken to complete execution also increases. Further, machine learning and

bioinformatics programs can consume a significant amount of time and resources to execute when used with typical data. Therefore, it becomes beneficial to prioritize MRs even with a smaller MR set size. Also often, individual MRs significantly differ in their fault detection effectiveness and some MRs tend to detect the same type of faults [77]. Therefore, selecting a diverse set of MRs enhances the fault detection effectiveness of MT [14, 18, 49] and would help to save resources during regression testing. However, while Cao et al. [14] work provides hints/quantitative guidelines towards this direction, there are no existing automated methods for selecting or prioritizing MRs based on their potential fault detection effectiveness.

3.2 Fault-based MR Prioritization

Let the current version of the SUT be v_k. Fault-based MR prioritization utilizes fault detection information of MRs collected while testing the previous version v_{k-1} of the SUT to prioritize MRs for testing v_k. The Fault-based prioritization approach involves the process of logging the faults revealed by each MR when conducting testing on previous versions of the SUT and prioritizing the MRs using that information. Results of mutation analysis conducted on previous versions of the SUT can also be utilized in this process. Figure 3.1 explains the process. Below, we describe the steps for creating the prioritized MR order:

1. Let the set of source test cases used for testing the version v_{k-1} of the SUT be the *prioritizing source test cases* (T_{sp}).

2. Let the set of follow-up test cases used for testing the version v_{k-1} of the SUT be the *prioritizing follow-up test cases* (T_{fp}).

3. Let the set of faults detected in version v_{k-1} be the *prioritizing set of faults* F_p. For each fault $f \in F_p$, log whether each MR_i revealed f when executed

with (T_{sp}) and (T_{fp}). Here, MR_i refers to metamorphic relations (MR_1, MR_2....MR_n) used for testing with i=1 to n.

4. Use the following greedy approach to create the prioritized ordering of the MRs:

   (a) Select the MR that revealed the highest number of faults in F_p and place it in the prioritized MR ordering. If there are multiple MRs with the same highest number, select one MR from them randomly.

   (b) Remove each $f \in F_p$ detected as faulty by that MR from F_p.

   (c) Repeat steps 4a and 4b until all the possible faults are revealed.

5. Select the top n MRs from the prioritized ordering to execute based on the resources available for testing.

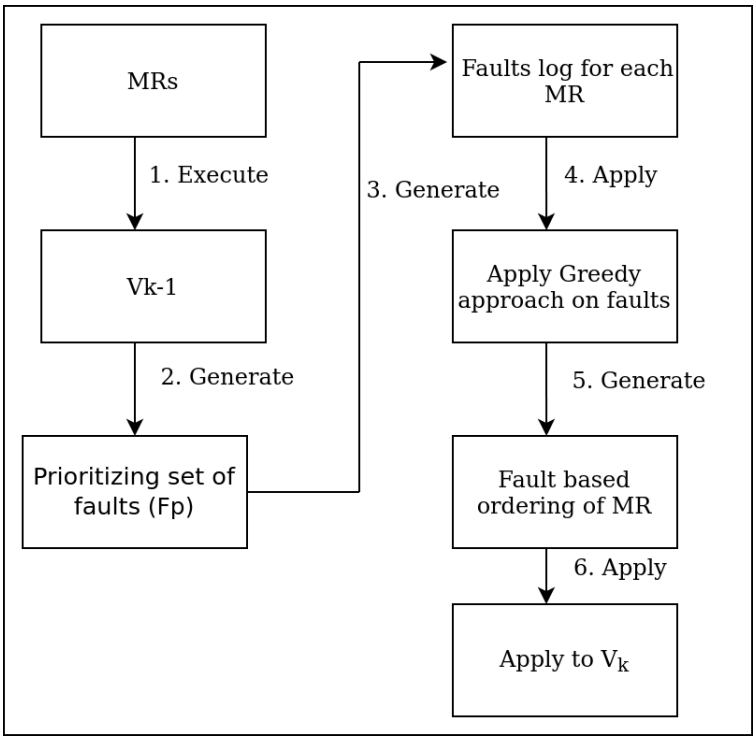


Figure 3.1: Fault-based MR Prioritization

3.3 Coverage-based MR Prioritization

Let the current version of the SUT be v_k. Coverage-based MR prioritization uses statement or branch coverage information collected when testing the previous version of the SUT for prioritizing MRs. Figure 3.2 explains the process. Below, we describe the steps for creating the prioritized MR order as follows:

1. Let the set of source test cases used for testing the version v_{k-1} of the SUT be the *prioritizing source test cases* (T_{sp}).

2. Let the set of follow-up test cases used for testing the version v_{k-1} of the SUT be the *prioritizing follow-up test cases* (T_{fp}).

3. For each MR_i, log the statements (or branches) that were executed when running the corresponding source and follow-up test cases from (T_{sp}) and (T_{fp}) on version v_{k-1} of the SUT. Here, MR_i refers to metamorphic relations (MR_1, $MR_2....MR_n$) used for testing with i=1 to n.

4. For each MR_i, compute the union of statements (or branches) executed by their corresponding source and follow-up cases.

5. Use the following greedy approach based on the statement (or branch) coverage to create the prioritized ordering of the MRs for testing version v_k.

   (a) Select the MR that covers the highest number of statements (or branches) and place it in the prioritized MR ordering. If there are multiple MRs with the same highest coverage, select one MR from them randomly.

   (b) Remove the statements (or branches) covered by that MR from further consideration.

(c) Repeat Steps 5a and 5b until all the possible statements (or branches) are covered.

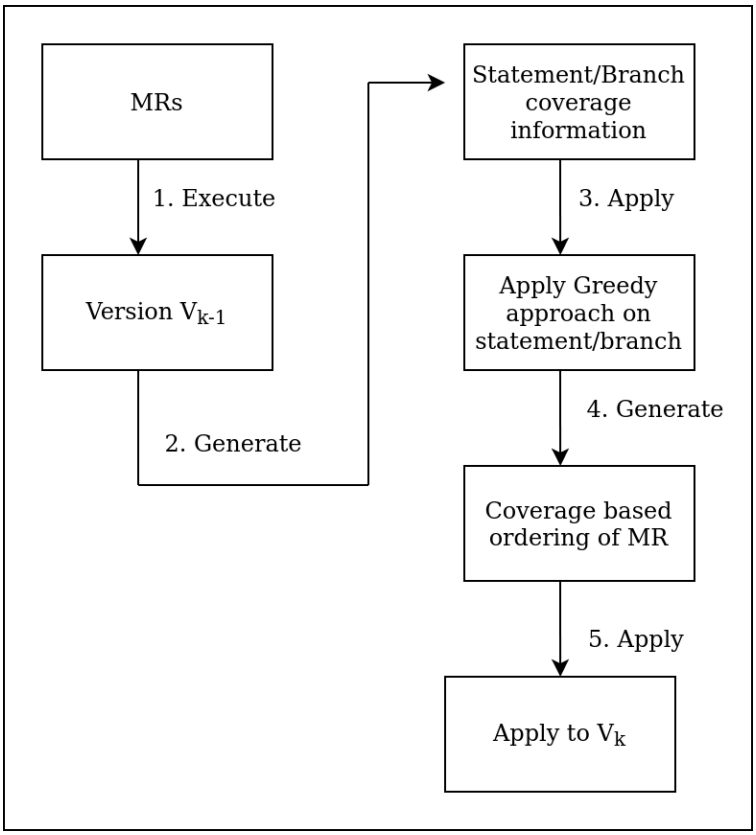


Figure 3.2: Coverage-based MR Prioritization

3.4 Evaluation

In this work, we plan to evaluate the utility of the developed prioritization approaches on the following aspects: (1) fault detection effectiveness of MR sets, (2) effective number of MRs required for testing, and (3) Time taken to detect a fault. To the best of our knowledge, there are no existing methods for MR prioritization. Therefore, to evaluate the effectiveness of the Fault-based and Coverage-based prioritizing approaches, we developed (1) a *random baseline*: this

represents the current practice of executing source and follow-up test cases of the MRs in random order. (2) an *Optimal ordering*: this represents the MR order that yields the maximum achievable fault detection effectiveness with the given set of MRs and source/follow-up test cases.

We conducted experiments to find answers for the following research questions:

1. Research Question 1 (RQ1): *Are the proposed MR prioritization approaches more effective than the random baseline?*

2. Research Question 2 (RQ2): *How do the proposed MR prioritization approaches perform compared to the Optimal ordering?*

3. Research Question 3 (RQ3): *How does the Fault-based prioritization approach perform against statement and branch coverage based prioritization approaches?*

3.4.1 Evaluation Procedure

In order to answer the above research questions, we carry out the following validation procedure:

1. We used mutants generated on version v_{k-1} of the SUT using a mutation engine to represent the faults in F_p described in Section 3.2. T_{sp} and T_{fp} are executed on F_p and any mutants that gave exceptions or mutants that did not finish execution within a reasonable amount of time were removed from this set.

2. We create a set of source test cases for the SUT independent of the T_{sp} used in Step 1 for validation. These source test cases will be referred to as the *validation source test cases* (T_{sv}). Then we create the follow-up test cases according to the MRs used for conducting MT on a given SUT. These will be referred to as the *validation follow-up test cases* (T_{fv}).

3. We generate a set of mutants for the version v_k of the SUT independent of the the F_p mentioned in Section 3.2. These mutants will be referred to as the *validation set of faults* (F_v). To generate F_v, we use a different mutation engine than the one used for generating F_p. Then from the generated mutants, we remove mutants that give exceptions or mutants that do not finish execution within a reasonable amount of time from further consideration. Also, we remove any mutants that are making the same syntactic change as the mutants in F_p from F_v.

4. We use the method described in Section 3.2 to obtain the Fault-based MR ordering. Then we applied the obtained MR ordering to F_v and logged the mutant killing information.

5. We use the method described in Section 3.3 to obtain the statement-coverage based MR ordering. Then we applied the obtained Coverage-based MR ordering to F_v and logged the mutant killing information of the MRs.

6. We used the method described in Section 3.3 to obtain the branch-coverage based MR ordering. Then we applied the obtained Coverage-based MR ordering to F_v and logged the mutants killing information of the MRs.

7. *Creating the random baseline*: We generate 100 random MR orderings and apply each of those orderings to F_v. The mutant killing information for each of those random orderings is logged. We generate 100 random MR orderings and apply each of those orderings to F_v. The mutant killing information for each of those random orderings is logged. The average mutant killing rates of these 100 random orderings were computed to get the fault detection effectiveness of the random baseline.

8. *Creating the Optimal ordering*: Generate by applying the greedy approach described in Section 3.2 to F_v as follows:

   (a) Select the MR that kills the largest number of mutants in F_v and place it in the Optimal ordering.

   (b) Remove the mutants killed by that MR from F_v.

   (c) Repeat the previous two steps until all the possible mutants are killed.

3.4.2 Evaluation Measures

We used the following measures to evaluate the effectiveness of the MR orderings generated by our MR prioritization approaches:

1. To measure the *fault detection effectiveness* of a given set of MRs, we use the percentage of mutants killed by those MRs.

2. To calculate the *effective MR set size* we used the following approach: typically, the fault detection effectiveness of MT increases as the number of MRs used for testing increases. However, after a certain number of MRs, the rate of increase in fault detection slows due to factors such as redundancy of MRs. Therefore when there is no significant increase in the fault detection between two MR sets of consecutive sizes of size m and $m + 1$, where the MR set of size $m + 1$ is created by adding one MR to the MR set of size m, the effective MR set size can be determined. That is, if the difference in fault detection effectiveness of MR set of size m and MR set of size $m+1$ is less than some threshold value, m would be the effective MR set size that should be used for testing. Determining this threshold value should be done considering the critical nature of the SUT. In this work, we used two threshold values of 5% and 2.5% as used in previous related work for determining the oracle data set size [28].

3. We used the following approach to find the *average time taken to detect a fault*: for each killable mutant m in F_v, we computed the time taken to kill the mutant (t_m) by computing the time taken to execute the source and follow-up test cases of the MRs in a given prioritized order until m is killed (here it is assumed that the source and follow-up test cases for each MR are executed sequentially, even though in practice it might be possible to execute them in parallel). Then the average time taken to detect a fault is computed using the following formula:

$$\frac{\sum t_m}{\#killable\,mutants\,in\,F_v}$$

3.4.3 Subject Programs and MRs

We applied the above-mentioned validation procedure on the following three applications from diverse domains for evaluating our proposed MR prioritization methods.

- BBMap[1]: performs global alignment of RNA and DNA sequence reads. The inputs to the program are a set of reads and a reference genome that is used as the alignment reference. The output is the mapping of reads to the reference genome.

- IBk[2]: K-nearest neighbours (KNN) classifier implementation in the Weka machine learning library [2]. The input to IBk is a training data set, and a test data set to represent in .arff format. The output is the classification predictions made on the instances in the test data set.

- LingPipe[3]: a broad tool kit for processing text using computational linguistics

[1] `https://jgi.doe.gov/data-and-tools/bbtools/bb-tools-user-guide/BBMap-guide/`
[2] `http://weka.sourceforge.net/doc.dev/weka/classifiers/lazy/IBk.html`
[3] `http://alias-i.com/`

and often used in bioinformatics for bio-entity recognition. In this work, we only use the "bio-entity recognition" functionality that focuses on extracting biomedical terms from text and assigning them to specific categories. Input to the bio-entity recognition problem is a set of bio-medical article and it produces extracted bio-entities as the output.

These three applications fall into the category of non-testable programs and represent the programs where MT can be effectively utilized to conduct regression testing.

3.4.4 Metamorphic Relations

To conduct MT on the above mentioned subject programs, we used a set of MRs developed in previous studies. For testing BBMap, we used eight MRs (listed in Appendix A.1.1) developed by Giannoulatou et al. based on the expected behaviour of short-read alignment software [29]. All of these MRs specify modifications to the reads supplied as the input, such as by shuffling the reads randomly, duplicating reads, and removing reads. We evaluated the fault detection effectiveness of these MRs on BBMap (see Figure 3.3a and Figure 3.3b) using mutation testing (details of mutation testing is provided in Section 3.4.6). The result indicates that for both versions of BBMap, MR2 (Mapped reads) reported the highest fault detection effectiveness followed by MR7 (reverse complement of reads). MR5 (quality score increase of reads) had the lowest fault detection effectiveness for both BBMap versions.

For conducting MT on IBk, we used 11 MRs developed by Xie et al. [87] (listed in Appendix A.1.2). These MRs are developed based on the user expectations of supervised classifiers. These MRs modify the training and testing data so that the predictions do not change between the source and follow-up test cases. We evaluated the fault detection effectiveness of these MRs on IBK using mutation testing

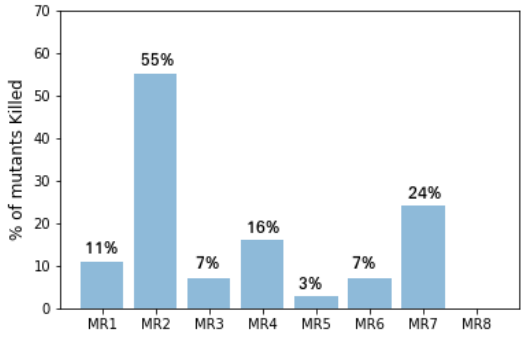
(see Figure 3.4a and Figure 3.4b). The result indicated that MR7 (permutation of class labels) and MR8 (Addition of informative attribute) provided the highest fault detection rate on both IBk versions. For version v_{k-1}, the lowest fault detection rate of 11% is reported by MR1 (consistence with affine transformation) and for version v_k the the lowest fault detection rate of 7% is reported by MR2 (permutation of the attribute)).

For testing the bio-entity recognition task in LingPipe, we used ten MRs that were developed in our previous work [77]. Appendix A.1.4 lists these MRs with additional constraints added to them to make them more generalizable. We describe three categories of MRs for testing bio-entity recognition software. These MR categories are addition, deletion, and shuffling. In addition relations, we extend the text by adding new text such that the sentence/paragraph boundaries are preserved. In the deletion relation, text is truncated by removing part of it such that the sentence boundaries are preserved. In the shuffling relations, we shuffle portions of the text. Figure 3.5a and Figure 3.5b shows the fault detection effectiveness (evaluated using mutation testing) of these MRs when used for testing the bio-entity recognition task in LingPipe. MR8 (Removing some words from a list of random word of an article) performed best in version $v_k - 1$ and v_k of the program. After that ordering of MRs according to the killing rates vary between the two mutant sets.

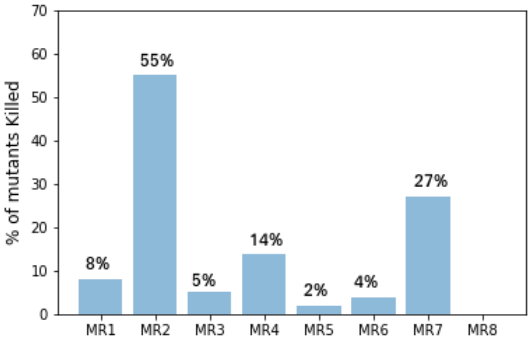
3.4.5 Source and Follow-up Test Cases

As we described before, MT involves the process of generating source and follow-up test cases based on the MRs used for testing. BBMap requires a set of reads and a reference genome as inputs. We used the E.coli reference genome and a set of its reads and Yeast reference genome and a set of its reads [4] as source test

[4] http://genome.ucsc.edu/



(a) Killing rate of MRs for v_{k-1} Mutants

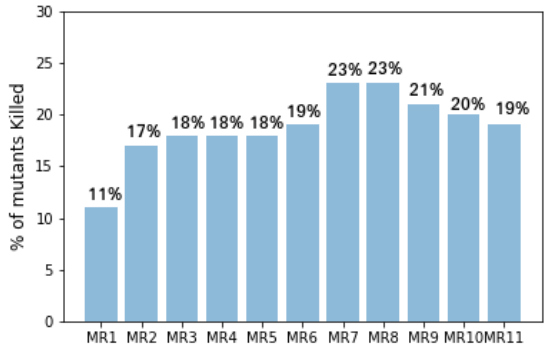


(b) Killing rate of MRs for v_k Mutants

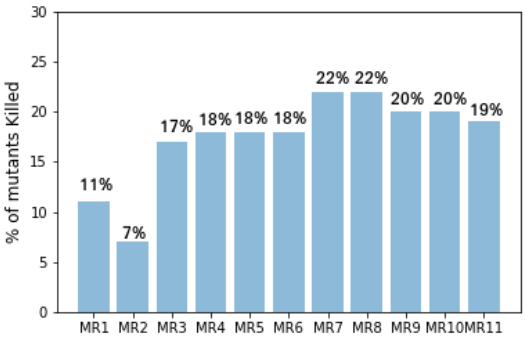
Figure 3.3: Fault detection effectiveness of individual MRs for BBMap Programs

cases. E.coli and Yeast are considered as model genomes and are widely used in the bioinformatics domain for conducting tests. To generate the follow-up test cases using these source test cases, we applied the input transformations described in the MRs listed in Appendix A.1.1.

IBk uses a training data set to train the k-nearest neighbor classifier, and a test data set is used to evaluate the performance of the trained classifier. Figure A.1 in Appendix A.1.2 shows an example dataset that is used as a source test case for IBk; the training data set is on the left and the test data set is on the right in the figure. After executing this source test case, the output will be the class labels predicted for the test data set, which is a value from the set {0,1,2,3,4} in this



(a) Killing rate of MRs for v_{k-1} Mutants



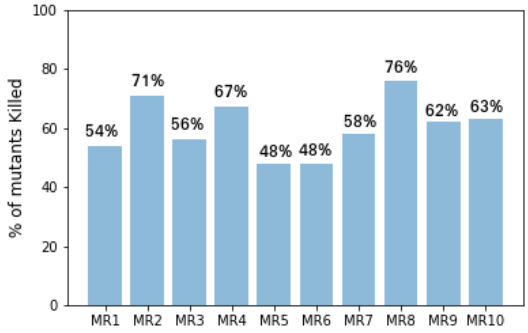
(b) Killing rate of MRs for v_k Mutants

Figure 3.4: Fault detection effectiveness of individual MRs for IBK Programs

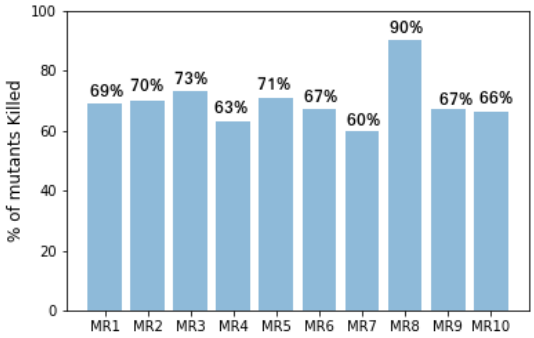
example. We generated ten similar datasets randomly, where each training and test data set contained four numerical attributes and a class label. The attribute values are randomly selected within the range [0, 100]. The values of the class label are randomly selected from the set {0,1,2,3,4,5}. The size of the training testing data sets ranges within [0, 200]. To generate the follow-up test cases using these source test cases, we applied the input transformations described in the MRs listed in Section A.1.2.

For creating source test cases for LingPipe, two biomedical articles with the PMCIDs 100320 and 100325 were obtained through PubMed [5]. The sentences and paragraphs used as source test cases for the MRs were picked randomly from these

[5]https://www.ncbi.nlm.nih.gov/pubmed



(a) Killing rate of MRs for v_{k-1} Mutants



(b) Killing rate of MRs for v_k Mutants

Figure 3.5: Fault detection effectiveness of individual MRs for LingPipe Program

two articles. However, for specific MRs related to removing some words from a list of random words (MR8) and shuffling a list of random words (MR10), 15 random articles were selected, and two sets of 500 random words were chosen from the selected articles to generate the source test case. To generate the follow-up test cases using these source test cases, we applied the input transformations described in the MRs listed in Section A.1.4.

3.4.6 Mutant Generation

For each version of the subject program, we aimed at developing two independently generated mutant sets to be used as F_p and F_v. For this, we used three

Table 3.1: Mutants generated for version v_{k-1} of the SUTs

Version	**Subject**	**#Major**	**#PIT**	**# μJava**
38.70	BBMap	102	100	N/A
3.8.5	IBk	1021	N/A	621
3.9.3	LingPipe	100	N/A	100

automated mutation tools: μJava[6], PIT[7] and Major[8]. For generating mutants using μJava, we used all the method level mutation operators [52]. With PIT mutation tool [23] and the Major mutation tool [41] we used all the available mutation operators provided by the tools. All the mutants the were generated using these tools were *first order mutants*, where a mutation operator is used to make a single modification to the source code to generate the mutant.

It is typical for mutation tools to generate some *equivalent mutants*; mutants that are syntactically different but functionally equivalent to the original program [5]. Therefore, these equivalent mutants always produce the same output as the original program and cannot be detected using testing. Due to the complexity of the subject programs and a large number of mutants generated, it was practically hard to find the equivalent mutants manually. Thus, we removed mutants that were giving the same output as the original program when executed with the above mentioned source test cases. Table 3.1 and Table 3.2 shows the number of mutants used for prioritization and evaluation after this filtering.

[6]https://github.com/jeffoffutt/muJava
[7]http://pitest.org/
[8]http://mutation-testing.org/

Table 3.2: Mutants generated for version v_k of the SUTs

Version	**Subject**	**#Major**	**#PIT**	**# μJava**
38.71	BBMap	102	100	N/A
3.8.6	IBk	1021	N/A	621
4.1.0	LingPipe	100	N/A	100

3.5 Results and Discussion

In this Section, we discuss our experimental results and provide answers to the three research questions that we listed in Section 3.4. For each subject program, we carried out the validation procedure described in Section 3.4 using the setup described in Table 3.3. In this setup, we used the generated mutant sets and the source test cases as F_p, T_{sp}, F_v, and T_{sv} in four different configurations. For example, the first evaluation run for BBMap was conducted by executing Ecoli as T_{sp}, Major mutants generated on version 38.70 (v_{k-1}) as F_p, Yeast as T_{sv}, and PIT mutants generated on version 38.71 (v_k) as F_v (refers to row 1 of Table 3.3). The second evaluation run for BBMap was conducted by executing Yeast as T_{sp}, PIT mutant generated on version 38.70 (v_{k-1}) as F_p, Ecoli as T_{sv}, and Major mutants generated on version 38.71 (v_k) as F_v (refers to row 2 of Table 3.3). Similarly, two additional evaluation runs were conducted for BBMap using the configurations in rows three and four in Table 3.3. In Table 3.3 for IBk, Test1 refers to 5 of the randomly generated datasets that we described in Section 3.4.5 and Test2 refers to the other five datasets. Figure 3.6 shows the average fault detection effectiveness for evaluation runs described above vs. the MR set size used for testing, for each subject program. We also plot the percentage of faults detected by the random baseline and the Optimal MR ordering for comparison.

Table 3.3: Validation Setup

Subject	$\mathbf{T_{sp}}$	$\mathbf{F_p}$ (SUT Version)	$\mathbf{T_{sv}}$	$\mathbf{F_v}$
BBMap	Ecoli	Major (38.70)	Yeast	PIT (38.71)
	Yeast	PIT (38.70)	Ecoli	Major (38.71)
	Ecoli	PIT (38.70)	Yeast	Major (38.71)
	Yeast	Major (38.70)	Ecoli	PIT (38.71)
IBk	Test1	Mujava (3.8.5)	Test2	Major (3.8.6)
	Test2	Major (3.8.5)	Test1	PIT (3.8.6)
	Test2	Mujava (3.8.5)	Test1	Major (3.8.6)
	Test1	Major (3.8.5)	Test2	Mujava (3.8.6)
LingPipe	ID:100325	Major (3.9.3)	ID:100320	PIT (4.1.0)
	ID:100320	Major (3.9.3)	ID:100325	PIT (4.1.0)
	ID:100320	PIT (3.9.3)	ID:100325	Major (4.1.0)
	ID:100325	PIT (3.9.3)	ID:100320	Major (4.1.0)

The prioritized MR orderings generated through this process for these three programs are available here.

3.5.1 RQ1: Comparison of the MR Prioritization approaches against the Random Baseline

3.5.1.1 Fault detection effectiveness We formulated the following statistical hypothesis to answer RQ1 in the context of fault detection effectiveness:

H_1: For a given MR set of size m, the fault detection effectiveness of the MR set produced by Fault-based approach is higher than that of the random baseline.

[8] https://github.com/MSU-STLab/MRPrioritization.git

H_2: For a given MR set of size m, the fault detection effectiveness of the MR set produced by statement coverage based approach is higher than that of the random baseline.

H_3: For a given MR set of size m, the fault detection effectiveness of the MR set produced by the branch coverage based approach is higher than that of the random baseline.

The null hypothesis H_{0x} for each of the above defined hypothesis H_X is that the Fault-based, statement coverage and branch Coverage-based approaches perform equal to or worse than the random baseline.

In Table 3.4, 3.5 and 3.6 we list the relative improvement in average fault detection effectiveness for the three MR prioritization methods over the currently used random approach for BBMap, IBk, and LingPipe, respectively. To evaluate the above hypotheses, we used the paired permutation test since it does not make any assumptions about the underlying distribution of the data [31]. We apply the paired permutation test to each MR set size for each of the subject programs with α= 0.05, and the relative improvements that are significant are marked with a *.

As shown in Table 3.4, for BBMap, the three prioritization approaches showed significant improvements in average fault detection effectiveness over the random approach for all the MR set sizes except for the last set size. Particularly, among the different MR set sizes, the increase in fault detection percentage varies from 3.70% to 266.66% between the three methods. For example, in a situation where only two MRs can be used for testing BBMap, using an MR prioritization would yield an increase of 11% - 122% in fault detection effectiveness. Therefore, we reject the null hypotheses H_{01}, H_{02}, and H_{03} for BBMap.

Table 3.5 shows the relative improvement in the average fault detection percent-

age for IBK where Fault-based MR prioritization shows significant improvements for all MR set sizes except for six, seven and last set size. However, statement and branch coverage-based methods did not show such consistent significant improvements in fault detection. Therefore we can only reject H_{01}, and we cannot reject H_{02} and H_{03} in general for IBK. This lack of performance of Coverage-based prioritization methods in IBK can be attributed to the considerable lack of variation in the statements/branches executed by the source and follow-up test cases of the MRs for IBK. This lead to placing only 3 out of the 11 MRs in the prioritized order based on the coverage and the other 7 were selected randomly based on the Coverage-based MR prioritization method described in Section 3.3. In fact, source and follow-up test cases of MR7 which killed the highest percentage of mutants in IBK (see Figure 3.4) covered the same statements/branches as MR1 which killed considerably low percentage of mutants. Since, an MR is picked randomly when the coverage is the same this lead to placing some of these MRs with low killing rates in the Coverage-based prioritized order first leading to reducing its performance. This lack of correlation between the coverage and fault detection is not surprising considering that IBK is a machine learning program that follows a different programming paradigm compared to traditional software [91].

Table 3.6 shows the relative improvement in the average fault detection percentage for LingPipe. While Fault-based MR prioritization shows improvements for MR set sizes $m = 1$ to $m = 4$, statement and branch coverage-based prioritization did not show such consistent improvements across different MR set sizes. Also overall, the relative improvement across MR prioritization methods varies between 0.23% - 11.53%, which is less than the other two subject programs. Further, for some MR sets, a random baseline slightly outperformed the statement and branch coverage-based prioritization. This is due to the fact that except for MR8, the statement and branch coverage of other MRs were similar for LingPipe. Therefore, Coverage-based

prioritization would yield in a similar performance as the random baseline since the MRs are picked randomly when the coverage is the same. Therefore we can only reject H_{01} for MR set sizes $m = 1$ to $m = 4$, and we cannot reject H_{02} and H_{03} in general for LingPipe.

Among the three programs, BBMap achieved the highest improvement in fault detection compared to the other two programs. This difference in the performance of can be explained using the difference in the fault detection effectiveness of individual MRs for BBMap compared to the other two subject programs. Figure 3.3, Figure 3.4 and Figure 3.5 shows the percentage of mutants killed by the MRs used for testing each subject program. As shown in Figure 3.5, MRs used for testing LingPipe has a higher average killing rate (70%) with a low variance among the killing rates (standard deviation = 8.15) and for IBk (see Figure 3.4), the average killing rate is 17.90% with a standard deviation of 4.78%. For MRs used for testing BBMap (see Figure 3.3), the average killing rate is 16.42% with a standard deviation of 18.41%. Since the variation in killing rates of the MRs was comparatively high in BBMap, it was able to gain a higher benefit by utilizing MR prioritization compared to IBk and LingPipe.

3.5.1.2 Effective number of MRs used for Testing In Figures 3.6a, 3.6b, and 3.6c, we show the effective number of MRs using the fault detection thresholds that we described in Section 3.4.2, for BBMap, IBk, and LingPipe, respectively. The purple vertical lines represent effective MR set size when using the Fault-based prioritization and the orange vertical lines represent the same for the random baseline. The respective thresholds are shown near each vertical line.

As shown in Figure 3.6a, for BBMap, the effective MR set size is 4 for the 5% threshold when Fault-based prioritization is used. With the random baseline, 5% threshold is never met. For IBk, (see Figure 3.6b), the effective MR set size with

Table 3.4: Relative improvement in fault detection effectiveness of the three MR prioritization approaches compared to random baseline for BBMap

MR Set-Size	**Fault-based**	**Statement Cov-based**	**Branch Cov-based**
1	266.66%*	40%*	40%*
2	122.22%*	11.11%*	11.11%*
3	68.42%*	2.63%*	2.63%*
4	54.34%*	4.34%*	4.34%*
5	31.48%*	3.70%*	3.70%*
6	16.39%*	0%	0%
7	5.97%*	-2.98%	-2.98%
8	0%	0%	0%

Fault-based prioritization is one for the 5% thresholds and for random baseline it is three. For LingPipe, the effective MR set size is two for the 2.5% threshold with Fault-based prioritization. With the random baseline, 2.5% thresholds are only met with MR set size 4. For LingPipe, the effective MR set size is three for the 5% threshold when branch coverage based prioritization and this threshold is never met for the random baseline. Therefore for all the three subject programs, using Fault-based prioritization resulted in a considerable reduction in the effective MR set size compared to using a random ordering of MRs. In practice, this reduction would translate into saving resources used for testing.

3.5.1.3 The average time taken to detect a fault Tables 3.9 and 3.10 shows the average time taken to detect a fault in BBMap and LingPipe respectively, for the four validation configurations described in Table 3.3. As shown in these results, using Fault-based prioritization resulted in significant reductions in the average time taken

Table 3.5: Relative improvement in fault detection effectiveness of the three MR prioritization approaches compared to random baseline for IBk

MR Set-Size	**Fault-based**	**Statement Cov-based**	**Branch Cov-based**
1	29.41%*	-11.76%	5.86%
2	9.52%*	-4.76%	-4.76%
3	4.34%*	0%	0%
4	4.16%*	0%	0%
5	4.0%*	0%	4%*
6	0%	3.84%	3.84%
7	0%	0%	0%
8	3.57%*	0%	3.57%*
9	3.44%*	3.44%*	6.89%*
10	3.33%*	3.33%*	3.33%*
11	0%	0%	0%

to detect a fault ranging from 42%-69% relative to the random baseline for BBMap and 10%-40% relative to the random baseline for LingPipe.

3.5.2 RQ2: Comparison of proposed MR Prioritization approaches against Optimal Ranking

In order to answer the second research question, we formulated the following statistical hypothesis:

H_4: For a given MR set of size m, the fault detection effectiveness of the MR set produced using Optimal ordering is higher than the fault detection effectiveness of the MR set produced by the Fault-based prioritization.

Table 3.6: Relative improvement in fault detection effectiveness of the three MR prioritization approaches compared to random baseline for LingPipe

MR Set-Size	**Fault-based**	**Statement Cov-based**	**Branch Cov-based**
1	11.53%*	-1.47%*	-1.47%*
2	6.81%*	-1.23%*	-1.23%*
3	3.33%*	-1.16%*	-1.16%*
4	1.09%*	-1.12%*	-1.12%*
5	0%	0%	0%
6	0%	0%	0%
7	0%	0%	0%
8	-1.07%	-1.07%	-1.07%
9	0%	0%	0.23%
10	0%	0%	0%

H_5: For a given MR set of size m, the fault detection effectiveness of the MR set produced using Optimal ordering is higher than the fault detection effectiveness of the MR set produced by the statement coverage-based prioritization.

H_6: For a given MR set of size m, the fault detection effectiveness of the MR set produced using Optimal ordering is higher than the fault detection effectiveness of the MR set produced by the branch coverage-based prioritization.

The null hypothesis H_{0x} for each of the above defined Hypotheses H_x is that the MR sets generated by the corresponding MR prioritization approach performs equal to or worse than the MR sets generated by the Optimal ordering in terms of fault detection effectiveness. Table 3.11, 3.12 and 3.13 show the relative improvement in fault detection effectiveness between the Optimal ordering and the MR prioritization

Table 3.7: Time taken by an MR to execute the Statement Centrality-based metrics for the various subjects

SUT	**Statement Affected**	**Statement Impacted**	**Fault Propogation**
BBMap	645s	695s	647s
LingPipe	100s	145s	120s
Linear Regression	180s	220s	200s
Naive Bayes	165s	210s	180s

Table 3.8: Time taken by an MR to execute the Variable-based metric for the various subjects

SUT	**Variable-based**
BBMap	690s
LingPipe	160s
Linear Regression	300s
Naive Bayes	345s

methods developed by us. The results that are significant at $\alpha = 0.05$ are shown with a $*$. As discussed above, Optimal ordering represents the best ordering of MRs if the faults are known in advance. Thus, it represents the upper bound of fault detection for a given set of MRs and source/follow-up test cases. Here we attempt to find how closely do our MR prioritization approach performs compared to this upper bound.

As shown in Table 3.11, 3.12, 3.13 Fault-based approach reported a 0% increase for the MR set of size $m = 1$ for both BBMap and IBk. This indicates that the Fault-based MR prioritization is able to identify the most effective MR from the given set

Table 3.9: Comparison of average time taken to detect a fault using Fault-based MR prioritization and Random baseline for BBMap

$\mathbf{T_{sv}}$	$\mathbf{F_v}$	**Fault-based**	**Random baseline**	**% Time reduction**
Yeast	PIT	125.82s	402.09s	69%
Ecoli	Major	69.22s	168.21s	59%
Yeast	Major	126.51s	220.14s	42%
Ecoli	PIT	32.87s	79.06s	58%

Table 3.10: Comparison of average time taken to detect a fault using Fault-based MR prioritization and Random baseline for LingPipe

$\mathbf{T_{sv}}$	$\mathbf{F_v}$	**Fault-based**	**Random baseline**	**% Time reduction**
ID:100325	PIT	76s	93s	23%
ID:100320	Major	125s	210s	40%
ID:100320	PIT	96s	160s	40%
ID:100325	Major	113s	126s	10%

of MRs and placed it first in the prioritized MR set for these programs. However, the Coverage-based approaches were not able to achieve this. In particular, for IBk all the MR sets generated using the Coverage-based MR prioritization approaches reported a significantly lower fault detection rate when compared to the Optimal ordering. As discussed before, for IBk, MR7 reported the highest fault detection effectiveness. However, this MR did not report the highest statement or branch coverage. Similar observation can be made for LingPipe as well. Thus, Coverage-based MR prioritization approaches failed to choose the MR with the highest fault

Table 3.11: Average relative improvement in fault detection effectiveness of Optimal approach over Fault-based, Statement and Branch coverage-based approach for BBMap

MR Set-Size	**Fault-based**	**Statement Cov-based**	**Branch Cov-based**
1	0%	161.90%*	161.90%*
2	8.33%*	116.66%*	116.66%*
3	9.37%*	79.48%*	79.48%*
4	0%	47.91%*	47.91%*
5	0%	26.78%*	26.78%*
6	0%	16.39%*	16.39%*
7	0%	9.23%*	9.23%*
8	0%	0%	0%

detection effectiveness to be placed first in the generated prioritized order. This is not surprising since IBk is a supervised machine learning classifier that creates a machine learning model from the training data. Lingpipe also uses a trained supervised classifier. For such programs only achieving high coverage in the source code is not enough for achieving a high fault detection effectiveness, since that would not essentially represent the ability to identify issues in the developed machine learning model. Therefore, for those types of programs, it would be more effective to use Fault-based MR prioritization that directly uses the fault detection capabilities of MRs for creating the prioritized MR ordering.

In general, the results show that while the MR sets produced by the Optimal ordering have comparatively higher fault detection percentages than the three prioritization methods, the gap between them is considerably small, in particular for the Fault-based prioritization approach. Therefore, we can conclude that the

Table 3.12: Average relative improvement in fault detection effectiveness of Optimal approach over Fault-based, Statement and Branch coverage-based approach for IBk

MR Set-Size	**Fault-based**	**Statement Cov-based**	**Branch Cov-based**
1	0%	46.66%*	22.22%*
2	17.39%*	35%*	35%*
3	20.83%*	26.08%*	26.08%*
4	20%*	25%*	25%*
5	19.23%*	24%*	19.23%*
6	19.23%*	14.81%*	14.81%*
7	14.81%*	14.81%*	14.81%*
8	6.89%*	10.71%*	6.89%*
9	3.33%*	3.33%*	0%
10	0%	0%	0%
11	0%	0%	0%

Fault-based MR prioritization is quite effective in producing an MR ordering that is comparably effective as the Optimal ordering.

3.5.3 RQ3: Comparison of the Fault-based MR prioritization approach against the Coverage-based prioritization approaches

In this research question, we evaluate whether the Fault-based ranking approach outperforms statement and branch coverage-based approaches. In order to answer the research question, we formulate the following hypotheses:

H_7: For a given MR set of size m, the fault detection effectiveness of the MR set produced using Fault-based prioritization is higher than the fault

Table 3.13: Average relative improvement in fault detection effectiveness of Optimal approach over Fault-based, Statement and Branch coverage-based approach for LingPipe

MR Set-Size	**Fault-based**	**Statement Cov-based**	**Branch Cov-based**
1	13.33%*	24.44%*	24.44%*
2	4.34%*	11.95%*	11.95%*
3	3.22%*	7.52%*	7.52%*
4	3.19%*	5.31%*	5.31%*
5	3.19%*	3.19%*	3.19%*
6	2.12%*	2.12%*	2.12%*
7	1.06%	1.06%	1.06%
8	1.06%	1.06%	1.0%
9	0%	0%	0%
10	0%	0%	0%

detection effectiveness of the MR set produced by the statement coverage-based prioritization.

H_8: For a given MR set of size m, the fault detection effectiveness of the MR set produced using Fault-based prioritization is higher than the fault detection effectiveness of the MR set produced by the branch coverage-based prioritization.

The null hypothesis H_{0x} for each of the above defined Hypotheses H_x is that the MR sets generated by the Fault-based prioritization method performs equal to or worse than the MR sets generated by the statement and branch coverage-based prioritization in terms of fault detection effectiveness.

Table 3.14: Average relative improvement in fault detection effectiveness using Fault-based approach over Statement and Branch coverage-based prioritization approaches for BBMap

MR Set-Size	**BBMap**	
	Statement Coverage	**Branch Coverage**
1	161.90%*	161.90%*
2	100.10%*	100.10%*
3	64.10%*	64.10%*
4	47.91%	47.91%
5	26.78%*	26.78%*
6	16.39%*	16.39%*
7	9.23%*	9.23%*
8	0%	0%

Table 3.14 and Table 3.15 shows the relative improvement in fault detection percentage between the MR sets generated by Fault-based prioritization approach and the MR sets generated by a statement/branch coverage based prioritization approaches for BBMap, IBk and LingPipe, respectively. For all the three programs, the majority of MR sets generated by Fault-based prioritization had significantly higher fault detection effectiveness compared to the MR sets generated by statement and branch coverage based prioritization. However, for IBk there are at least one MR sizes where the fault detection effective reduces. Therefore, in general, we cannot reject the null hypotheses H_{07} and H_{08} for IBk.

In practice, these increases in improvements in fault detection effectiveness translate to potentially detecting more faults using Fault-based MR prioritization compared to Coverage-based MR prioritization methods, when there are restrictions

Table 3.15: Average relative improvement in fault detection effectiveness using Fault-based approach over Statement and Branch coverage-based prioritization approaches for IBk and LingPipe.

MR Set-Size	**IBk**		**LingPipe**	
	Stmt Cov	**Branch Cov**	**Stmt Cov**	**Branch Cov**
1	46.66%*	22.22%*	12.82%*	12.82%*
2	15%*	15%*	7.95%*	7.95%*
3	4.34%*	4.34%*	4.44%*	4.44%*
4	4.16%*	4.16%*	2.19%*	2.19%*
5	4%*	0%	0%	0%
6	-3.73%*	-3.72%*	0%	0%
7	0%	0%	0%	0%
8	3.57%*	0%	0%	0%
9	0%	-3.22%*	0%	0%
10	0%	0%	0%	0%
11	0%	0%		

on the number of MRs that could be executed due to time/budget constraints.

3.6 Threat to Validity

External Validity: We only used three subject programs for our validation. However, these three applications are complex systems that are from different domains. We believe that they are good representations of applications that face the oracle problem for which MT is effective and widely used. Thus, we believe that our results are generalizable to applications for which MT is applicable. We have generated approximately 200 mutants for each subject, where approximately

100 mutants used for generating the prioritized MR orderings and approximately 100 mutants are used for validation. These numbers were determined after considering the time and cost taken for executing them. However, it is possible that the number of mutants used is low.

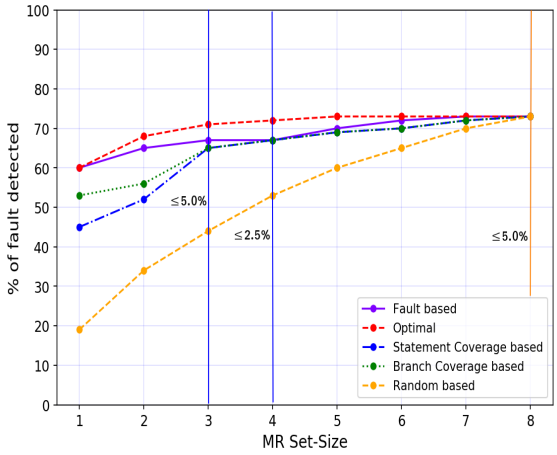
Internal Validity: For BBMap and LingPipe, we used existing test cases as source test cases. For IBk, source test cases were randomly generated. Generating source test cases using structural coverage criteria such as MC/DC (Modified Condition/ Decision coverage), Decision coverage based test cases might influence the fault detection ability of the MRs and hence result in a different MR ordering. The greedy approach used in this work to create the prioritized MR orderings selects an MR randomly when multiple MRs detect the same number of faults or if their corresponding source and follow-up test case executions have the same coverage. This could impact the effectiveness of the prioritized MR orderings. We plan to handle the limitation in our future work by incorporating metrics based on static and dynamic execution properties of the source and follow-up test cases to prioritize MRs. In a situation where the MRs would change from one version to another, the prioritized MR ordering has to be recomputed. However, in practice this not something that would happen often, since the MRs are necessary properties of SUT that are derived from the overall functionality or the underlying algorithm of the SUT [19].

Construct Validity: We measure the fault finding effectiveness of the MRs using mutants rather than real faults. MR prioritization carried out based on real faults might lead to a different result. However, studies conducted out by Just et al. [42] have shown a significant correlation between the detection of seeded faults and the detection of real faults.

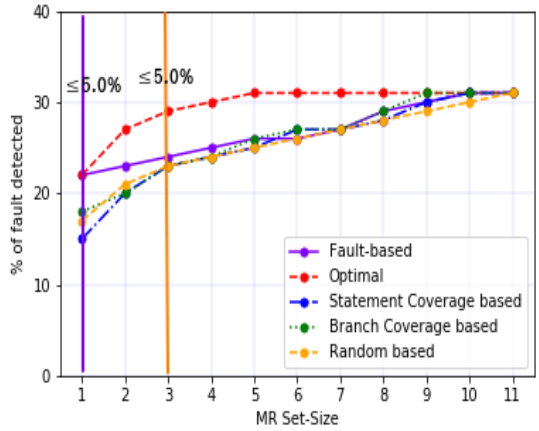
3.7 Summary

In this work, we developed automated approaches for MR prioritization in order to improve the efficiency and effectiveness of MT. These approaches use (1) fault detection information, and (2) statement/branch coverage information to prioritize MRs. We conducted an empirical study to evaluate the effectiveness of the developed approaches using three complex open-source software systems.

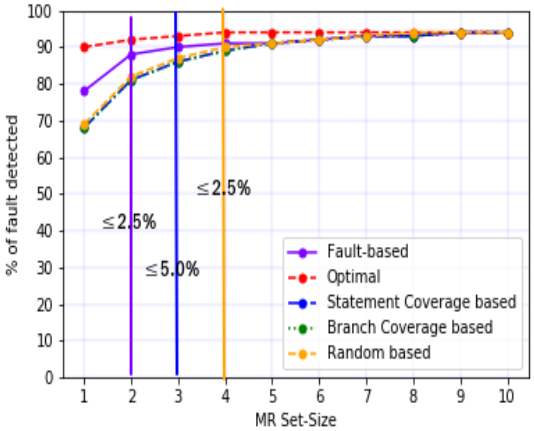
The experimental results show that utilizing MR prioritization would increase the fault detection effectiveness of a given MR set and these increases could range up to 266.66%. In particular, using Fault-based MR prioritization reduced the number of MRs that needs to be executed during testing and also reduced the average time taken to detect a fault compared to the currently used method of executing the source and follow-up test cases of the MRs in random order. Further, our results show that Fault-based MR prioritization outperforms the Coverage-based MR prioritization and performs comparably to the Optimal MR ordering. Based on these results, we recommend utilizing MR prioritization to improve the efficiency and effectiveness of MT.



(a) BBMap



(b) IBk



(c) LingPipe

Figure 3.6: Fault detection effectiveness for BBMap, IBk and LingPipe

CHAPTER FOUR

STATEMENT CENTRALITY AND VARIABLE-BASED APPROACH

In this Chapter, we cover two MR prioritization approaches to overcome some of the challenges faced by fault and coverage based approach. The Chapter is organized as follows: Section 4.1 provides motivation for Statement Centrality and Variable-based approach. Section 4.2 and 4.3 provides the proposed approaches for MR prioritization. Section 4.4 provides the evaluation procedure. Section 4.6 provides the results and answers to the research questions. Section 4.7 provides potential threats to validity. Section 4.8 provides discussion of conclusions.

4.1 Motivation

In the previous Chapter 3, we presented a fault and Code Coverage-based approach for prioritizing MRs in the context of regression testing. First, the results from applying the Code Coverage-based approach indicates that the fault detection effectiveness of the Code Coverage-based approach was equal to the random-based approach for all the subject programs except BBMap. Also, the approach did not work effectively for a machine learning program because achieving high coverage in the source code is not enough for identifying fault detection effectiveness, since that would not essentially represent the ability to identify issues in the developed machine learning model. Second, the Fault-based approach becomes costly when software testers have limited time to conduct metamorphic testing since the approach requires generating and executing a large number of mutants to prioritize MRs. So, we proposed a statement and Variable-based approach to overcome these challenges.

Figure 4.1 provides a motivating example for Statement Centrality-based

approach. The example in Figure 4.1 shows the difference in the Code Coverage between the source and follow-up test case executions for an MR, where, line 8, 9, and 10 is not executed by the source test case, but are executed by the follow-up test case. Cao et al. [14] quantitatively identifies dissimilarity between the code coverage of the source and follow-up test cases to select MRs, where the dissimilarity is computed by calculating the distance between the execution profiles of the initial and follow-up test cases. However, when the code coverage of the source and follow-up test cases are analyzed qualitatively, lines 6, 18, and 19 executed by the source test case do not contribute to the program output, i.e., the return statement in line 23. Therefore, it is important to compute the dissimilarity based on the importance and contribution of the statement to the program output.

Hence, we proposed a Statement Centrality method to qualitatively analyze the dissimilarity in the execution profiles of source and follow-up test cases by assigning weights to statements/branches based on factors such as statements affected, projected impact of the statement, and projected fault propagation. These factors can help in determining the importance of the statement to program execution when computing the dissimilarity in the execution profiles of source and follow-up test cases for an MR. Figure 4.2 explains the proposed approach.

Similarly, for the Variable-based approach, results from our previous work [76] indicated that code coverage is not effective in prioritizing MRs. Also, many times faults that occur inside the system do not propagate to the program output and violation of MR is not detected. Therefore, we proposed a Variable-based approach that uses internal variables in the program to prioritize MRs.

4.2 Statement Centrality Approach

In this approach, we propose three metrics to qualitatively analyze the dissimilarity (difference) in the execution profile of the source and follow-up test cases for an MR. The execution profile includes information about program elements such as statements, branches, and def-use pairs that were executed by the test cases. The intuition is that higher the dissimilarity in the execution profile between the test cases of an MR, then higher the chance of detecting failure [18].

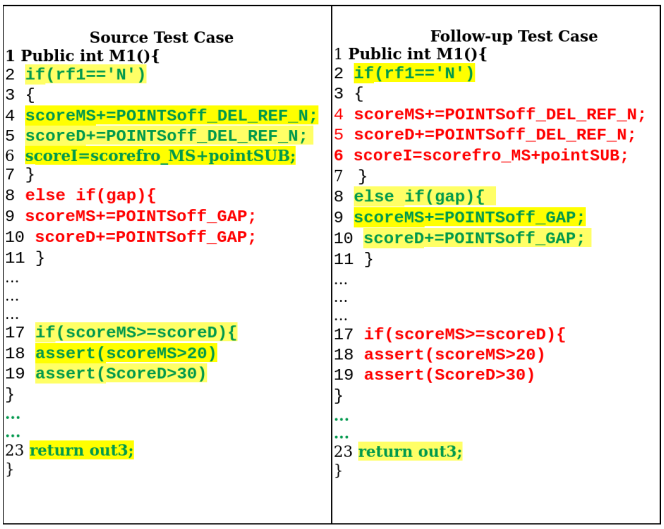


Figure 4.1: Motivating Example for Statement Centrality Method. Left shows the source test case execution and right shows the follow-up test case execution for an MR. The statements covered by the test cases are highlighted in yellow and uncovered statements are highlighted in red.

Let the current version of the SUT be v_k. In this approach, we apply the proposed metrics on version v_{k+1} and use that to prioritize the MRs for testing v_k as follows:

1. Let the set of source test cases used for testing the version v_k of the SUT be the *prioritizing source test cases* (T_{sp}).

2. Let the set of follow-up test cases used for testing the version v_k of the SUT be the *prioritizing follow-up test cases* (T_{fp}).

3. For each MR_i, log the statements (or branches) that were executed when running the corresponding source and follow-up test cases from (T_{sp}) and (T_{fp}) on version v_k of the SUT.

4. For each MR_i, compute the union of statements (or branches) executed by their corresponding source and follow-up cases. Let the union of statements be U_s.

5. For version v_k, compute backward slice from the return statement of a method. Let the set of data and control dependent statements in backward slice be B_r.

6. For each statement $s \in B_r$, identify whether $s \in U_s$. If $s \notin U_s$, then remove s from B_r.

7. For each of the statement $s \in B_r$, we calculate the weight for s by applying the proposed metrics: statement affected, projected impact of the statement, and projected fault propagation. The metrics are discussed in detail in Section 4.2.1 to 4.2.3.

8. Select the top n MRs from the prioritized ordering to execute based on the resources available for testing.

9. Apply the prioritized MR order to test v_{k+1}.

The following subsections will explain each of the proposed metrics in detail.

4.2.1 Statements Affected

The test execution can reach a faulty statement. The faulty statement causes the program to reach an incorrect program state, which is known as *infection* [5]. The

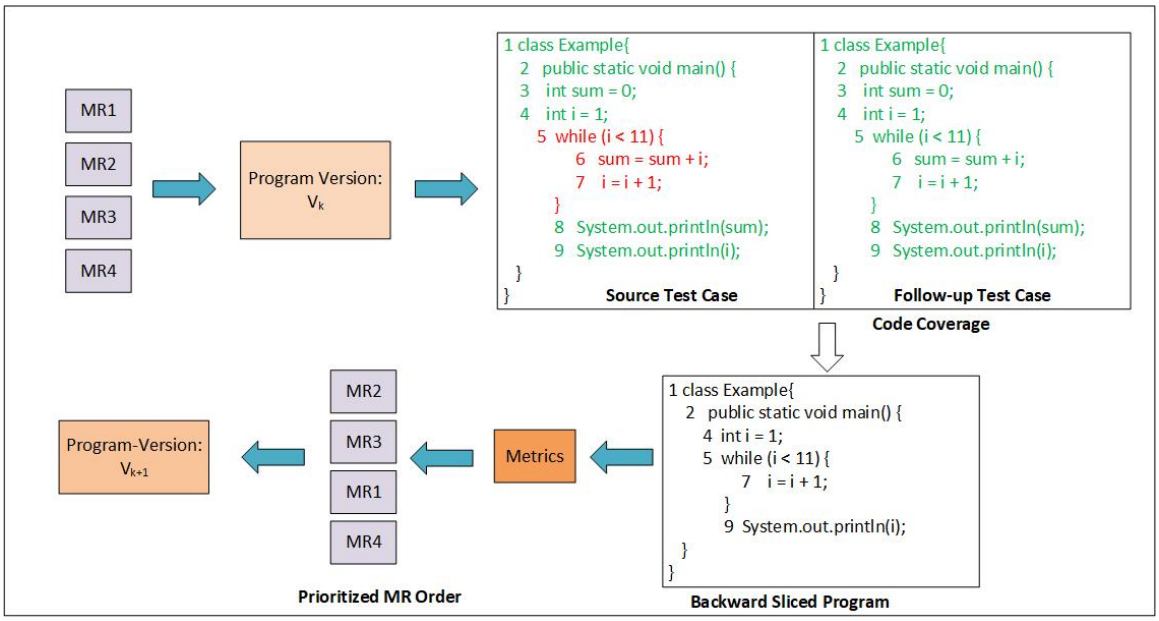


Figure 4.2: Statement Centrality Approach

infection in the statement propagates to all its data and control dependent statements; those statements are considered as *affected statements*. In Figure 4.3, infection in statement 1 could affect statement 3, 4 and 5 due to a fault. To identify the affected statements for a statement s, we apply the following steps:

1. For each statement $s \in B_r$

   (a) Apply forward slicing to s to identify the affected statements. Let the set of statements in the forward slice of statement s be F_s.

   (b) For each statement $s \in F_s$, identify whether $s \in U_s$. If $s \notin U_s$, then remove s from F_s.

   (c) The total statements affected for a statement s (SA_s) , which is given by the formula:

$$SA_s = \mid F_s \mid \tag{4.1}$$

where F_s is the set of statements in forward slice.

(d) Total statement affected for an MR (TA_s) is given by the formula:

$$TA_s = \sum_{s \in B_r} SA_s \tag{4.2}$$

where B_r is the set of statements in backward slice.

For example, consider the calculation of statements affected using the example in Figure 4.3. Total statements in the backward slice (B_r) is 10. Applying formula 4.1 to statement 1, SA_s is 5. The formula could be applied to all the statements in the slice. Similarly, applying formula 4.2 to get the total statement affected for an MR. The total statement affected for an MR is 13.

4.2.2 Projected Impact of a Statement

The affected statements can propagate the infection to all its data and control dependent statements. As a result, the affected statement can impact or create an effect on all its dependent statements. Those statements are called *impacted statement*. For example, in Figure 4.4, the affected node three in the CFG impacts node six due to an infection in node one, which is shown in the dotted line. Since node three impacts node six, the projected impact of node three is one. The projected impact of the statement is calculated using the following steps:

1. For each $s \in B_r$

   (a) Compute the forward slice of s, F_s. This provides the affected statements.

   (b) For each statement s′ in F_s

      i. Compute the forward slice $F_{s'}$. This gives the projected impacted statement.

ii. For each statement $s \in F_{s'}$, identify whether $s \in U_s$. If $s \notin U_s$, then remove s from $F_{s'}$.

iii. The total projected impact of statement ($SI_{s'}$) for a statement s′ is given by the formula:

$$SI_{s'} = \mid F_{s'} \mid \tag{4.3}$$

iv. The total projected impact of statement (TI_s) for an MR is given by the formula:

$$TI_s = \sum_{s' \in F_s} SI_{s'} \tag{4.4}$$

For example consider the calculation of projected impacted statements based on the example in Figure 4.4. The total statements in the backward slice (B_r) is 10. For statement 1 in the backward slice, the total projected statement impacted is calculated below.

1. For statement 1 in the backward slice, apply the step1 to get the statement affected. The forward slice of statement 1 (F_s) is statements 3, 4, and 5.

2. Apply step2, where we apply forward slice ($F_{s'}$) to get the projected impact for statements 3, 4, and 5.

   (a) Apply formula 4.3, to get the projected impact for statement 3. $SI_{s'}$ is 1.

   (b) Similarly, the projected impact for statement 4 ($SI_{s'}$) is 1.

   (c) Similarly, the projected impact for statement 5 ($SI_{s'}$) is 1.

   (d) The steps are carried out for all the statements in the backward slice.

(e) Apply formula 4.4, to get the total projected impact for an MR. The total projected impact for the MR (TI_s) is 27.

4.2.3 Projected Fault Propagation

The source and follow-up test cases in an MR should execute the faulty statement and the infection caused by the fault must propagate to the output. The possibility of infection reaching the output depends on two factors: (1) the distance between the statement in the program to the output and (2) the total number of operations in the statement. In our work, the return statement is considered the output statement, since the result of a specific task done by a method is passed to the return statement.

In this metric, we use the two factors mentioned earlier to calculate the fault propagation ability of statements in the backward slice. The distance is greater between the statement and output; then, it is less likely that the infection would reach the output statement. The number of operations in each statement in the backward slice is taken into consideration since more arithmetic and conditional operations in a statement increase the likelihood of error in the program. The steps for calculating the fault propagation from an impacted statement are provided below.

1. For each statement $s \in B_r$

   (a) *Generate abstract syntax tree* (AST) for s. The AST is used to count the number of arithmetic or conditional operators in s.

   (b) Traverse from s to the return statement in the control flow graph (CFG) to get the distance between s to output.

   (c) We calculate the projected fault propagation from statement s (PF_s) to

the output using the formula.

$$PF_s = 1/(\text{Total operations in s} + \text{Distance between s and output}) \quad (4.5)$$

Where,

Total Operations in s= Total Operators in AST for s,

Distance between s and output= Number of hops from s to the output node (return statement) in CFG.

(d) We calculate the total fault propagation (TFP) for an MR using the formula.

$$TFP = \sum_{s \in B_r} PF_s \quad (4.6)$$

For example consider the calculation of projected fault propagation based on the example in Figure 4.4. Statement 3 impacts statement 6. The number of operations in statement 6 is 5. The distance between statement 6 and output statement 10 is 4. We apply formula 4.5 to get the projected fault propagation for statement 6. The projected fault propagation (PF_s) for statement 6 is 0.1. Total fault propagation (TFP) for an MR using formula 4.6 is 10.6.

4.2.4 MR Quality Score

The proposed metrics indicate the extent of dissimilarity between the execution profile of the test cases and help in determining the MR quality score. The MRs are prioritized using the MR quality score. The score is based on the total statements affected, the total projected impact of statements, and the projected fault

propagation. The quality score for an MR is calculated using the formula:

$$\begin{aligned} \text{MR Quality Score} &= \text{Total Statements affected } (TA_s) \\ &+ \text{Total Projected Impact of Statement } (TI_s) \\ &+ \text{Total Projected Fault Propagation (TFP).} \end{aligned} \tag{4.7}$$

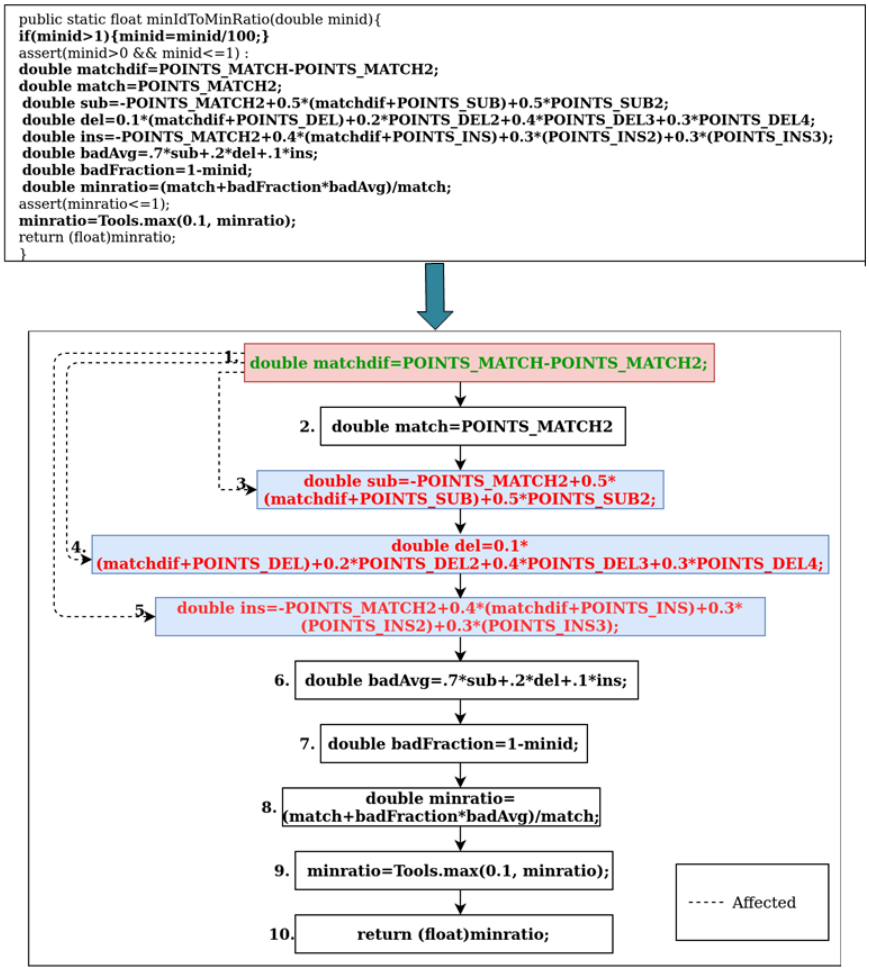


Figure 4.3: Example to show statements affected

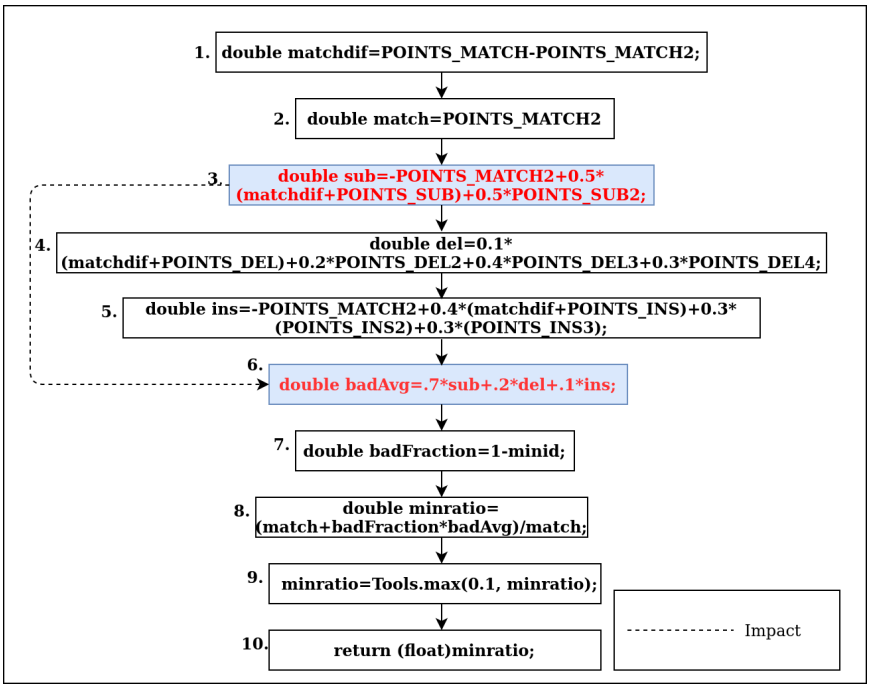


Figure 4.4: Statement Impacted due to a affected statement

4.3 Variable-based Approach

In this approach, we apply our proposed metrics to the set of internal and output variables triggered by the execution of source and follow-up test cases of an MR. The main intuition behind the approach is that the source and follow-up test cases of an MR should traverse the different paths to detect failure; hence, internal and output variables values should also exhibit dissimilar values.

Figure 4.5 provides a motivation for the Variable-based approach, where, in the example program, the source and follow-up test cases cover the same set of statements and do not show dissimilarity between the test cases for the MR. As a result, the existing test case dissimilarity-based MR selection approach proposed by Cao et al. [14] cannot be applied in this context. However, the dissimilarity between the source and follow-up test case execution is observed in the internal and

output variables. For instance, the example provided in Figure 4.5 shows statements 12 and 14 are executed by the source and follow-up test case, but the variable minscoreoff and floor contain a dissimilar value. Therefore, we proposed the Variable-based approach to qualitatively compute dissimilarity based on internal and output variables. Figure 4.6 explains the proposed approach.

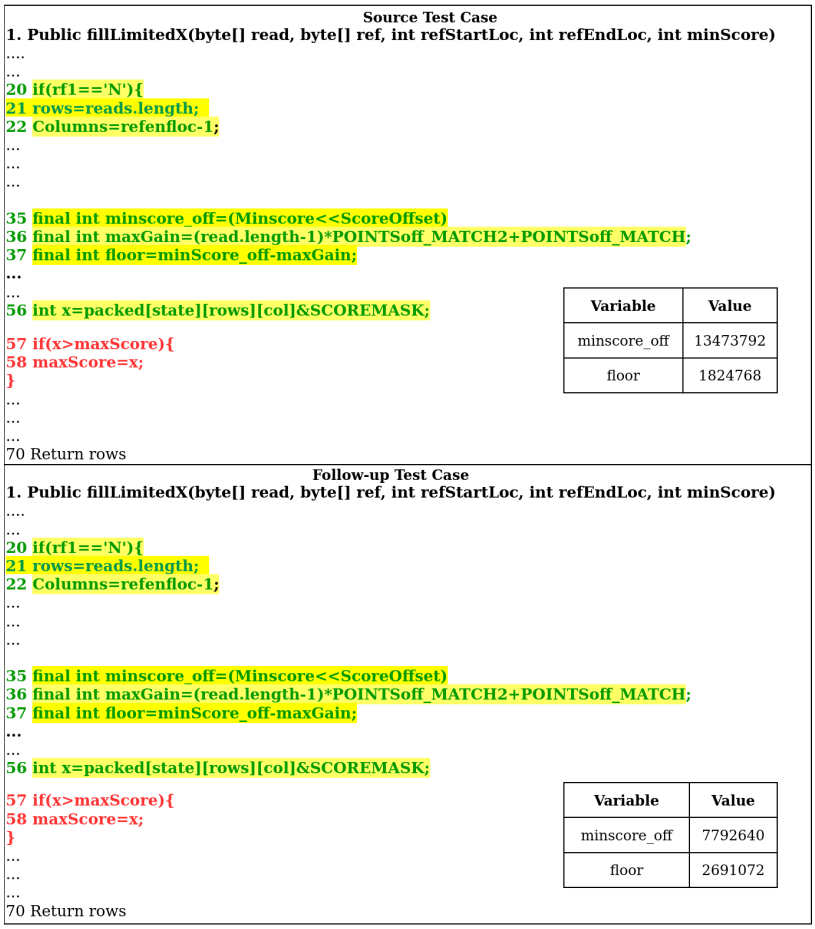


Figure 4.5: Motivating Example for Variable-based Method. Top shows the source test case execution and bottom shows the follow-up test case execution for an MR. The statements executed by the test cases are highlighted in yellow and statement not covered by the test cases are highlighted in red.

Let the current version of the SUT be v_k. In this approach, we apply the proposed metric on v_{k+1} and use that to prioritize the MRs for testing v_k as follows:

1. Let the set of source test cases used for testing the version v_k of the SUT be the *prioritizing source test cases* (T_{sp}).

2. Let the set of follow-up test cases used for testing the version v_k of the SUT be the *prioritizing follow-up test cases* (T_{fp}).

3. Select the internal variables to be monitored on v_k using the *variable selection* method described in Section 4.3.1. The output variables are the variables in the return statement of a method.

4. Execute the corresponding source and follow-up test cases from (T_{sp}) and (T_{fp}) on version v_k of the SUT.

5. For each MR_i, log the values of the selected internal variable.

6. Process the log information containing the values of the selected internal and output variables and apply the metrics. The metric is explained in Section 4.3.2.

7. Order the MRs based on the MR quality score.

8. Apply the prioritized MR order on $v_k + 1$.

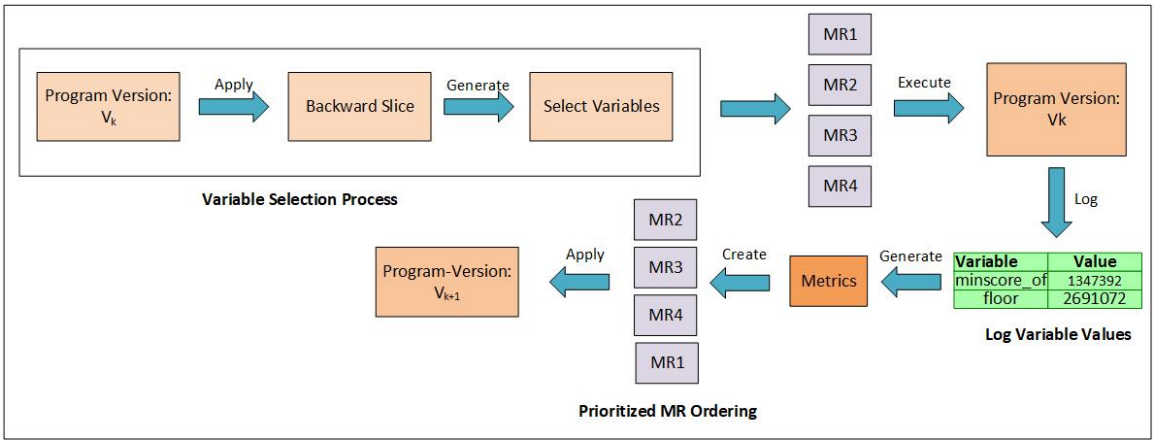


Figure 4.6: Variable-based Prioritization Approach

4.3.1 Variable Selection

Source code contains many variables, and certain variables have zero or minimum impact on the output, hence it is important to select variables carefully for monitoring. To select the variables for inspection, we apply the following steps:

1. For version v_k, apply backward slice from the return statement of a method. Let the set of data and control dependent statements in backward slice be B_s.

2. For each of the sliced statement $s \in B_s$, we calculate the weight by applying the statement affected metric presented in Section 4.2.1.

3. Order the statements in B_s in the decreasing order of weight. Higher the weight shows greater importance of the statement to the output.

4. Select the definitions variable from each statement in B_s and generate a set of final candidate variables C_v.

5. Apply the variable value dissimilarity metric discussed in Section 4.3.2 to C_v and prioritize the MRs.

We propose a metric for prioritizing MR. The metric is discussed below:

4.3.2 Variable Value Dissimilarity

Here, we hypothesize that good MRs should have different execution paths for source and follow-up test cases as shown in Chen et al. [18] work; therefore, internal variables should have different values. The steps for identifying dissimilarity in the variable value is explained below:

1. For each MR_i, execute the corresponding source and follow-up test cases from (T_{sp}) and (T_{fp}) on version v_{k-1} of the SUT.

2. For each variable $v \in C_v$.

   (a) Log the values of variable v_s and v_f. Let v_s and v_f be the value of variable for source and follow-up test case execution.

   (b) Verify whether value of v_s and v_f is different. The dissimilarity in the value between v_s and v_f indicates the diversity in the execution between the source and follow-up test cases for an MR.

   (c) v is considered killed when the dissimilarity in the value between v_s and v_f is detected.

   We calculate the total dissimilarity in the variable value (TD_V) for an MR using the formula below:

$$TD_V = \mid K_v \mid \tag{4.8}$$

   Where K_v represents the set of killed variable.

   Calculation:

We provide the calculation based on an example provided in Figure 4.5. The example code is taken from BBMap program. Table 4.1 provides a list of variables selected and kill/alive information for each variable. The variables were selected by applying the statement affected metric presented in Section 4.2.1. The metamorphic relation used in the example is mapped MR, where the source test case is an E. coli read and follow-up test case is a mapped E. coli read.

We apply formula 4.8 and the total dissimilarity in the variable value (TD_V) is 1 since one variable is killed.

Table 4.1: Variable Kill/Alive information based on the example program for an MR

Variables	**Source Test Case Value**	**Follow-up Test Case Value**	**Kill/Alive**
matchdiff	56717312	56717312	Alive
match	9967616	9967616	Alive
sub	1	1	Alive
del	5480448	825344	Kill
Ins	5480448	5480448	Alive
badAvg	1069056	1069056	Alive
badFraction	2	2	Alive
minratio	28062	28062	Alive

4.3.3 MR Quality Score

The MRs are prioritized using the MR quality score. The score is based on the total dissimilarity in the variable value. The quality score for an MR is calculated

using the formula:

$$\text{MR Quality Score} = \text{Total dissimilarity in the variable value } (TD_V) \tag{4.9}$$

4.4 Evaluation

In this work, we plan to evaluate the utility of the developed prioritization approaches similar to our previous work [76] and on the following aspects: (1) fault detection effectiveness of MR sets, (2) effective number of MRs required for testing, (3) time taken to detect a fault and (4) average percentage of faults detected. We evaluate the effectiveness of the Statement Centrality-based and Variable-based prioritizing approaches with (1) a *random baseline*: this represents the current practice of executing source and follow-up test cases of the MRs in random order. (2) An *Optimal ordering*: this represents the MR order that yields the maximum achievable fault detection effectiveness with the given set of MRs and source/follow-up test cases as proposed in our previous work [76].(3) *Fault-based ordering*: this represents the MR order based on the fault detection capability of the MRs. The approach selects an MR which has killed the highest number of mutants and places it in the prioritized MR ordering. The prioritization process continues until all the MRs are prioritized and faults have been exposed [76]. 4) *Code Coverage-based ordering*: this represents the MR order based on the statement and branch-coverage information of the MRs. The approach selects the MR which has covered the highest statements or branches in the SUT and places it in the prioritized MR ordering. The prioritization continues until all the statements and branches are covered [76]. (5) *Test case dissimilarity based ordering*: the approach selects MR by calculating the distance between the source and follow-up test cases proposed by Cao et al. [14].

We conducted experiments to find answers to the following research questions:

1. Research Question 1 (RQ1): *Are the proposed MR prioritization approaches more effective than the random baseline?*

2. Research Question 2 (RQ2): *How do the proposed MR prioritization approaches perform compared to the Optimal ordering?*

3. Research Question 3 (RQ3): *How do the Statement Centrality and Variable-based approaches perform against statement and branch coverage-based prioritization approaches?*

4. Research Question 4 (RQ4): *How do the Statement Centrality and Variable-based approaches perform against the Fault-based approach?*

4.4.1 Evaluation Procedure

In order to answer the above research questions, we carry out the following validation procedure similar to the previous work [76]:

1. We create a set of source test cases for the SUT independent of the T_{sp} for validation. These source test cases will be referred to as the *validation source test cases* (T_{sv}). Then we create the follow-up test cases according to the MRs used for conducting MT on a given SUT. These will be referred to as the *validation follow-up test cases* (T_{fv}).

2. We generate a set of mutants for the version v_k of the SUT. These mutants will be referred to as the *validation set of faults* (F_v). Then from the generated mutants, we remove mutants that give exceptions or mutants that do not finish execution from further consideration.

3. We use the method described in Section 4.2 to obtain Statement Centrality-based MR ordering. Then we applied the obtained MR ordering to F_v and logged the mutant killing information.

4. We use the method described in Section 4.3 to obtain the Variable-based MR ordering. Then we applied the obtained MR ordering to F_v and logged the mutant killing information of the MRs.

5. *Creating the random baseline*: We generate 100 random MR orderings and apply each of those orderings to F_v. The mutant killing information for each of those random orderings is logged. The average mutant killing rates of these 100 random orderings were computed to get the fault detection effectiveness of the random baseline.

6. *Creating the Optimal ordering*: Generate by applying the greedy approach as proposed in our previous work [76] to F_v as follows:

   (a) Select the MR that kills the largest number of mutants in F_v and place it in the Optimal ordering.

   (b) Remove the mutants killed by that MR from F_v.

   (c) Repeat the previous two steps until all the possible mutants are killed.

4.4.2 Evaluation Measures

We used the following measures to evaluate the effectiveness of the MR orderings generated by our MR prioritization approaches:

1. To measure the *fault detection effectiveness* of a given set of MRs, we use the percentage of mutants killed by those MRs.

2. To calculate the *effective MR set size* we used the following approach: typically, the fault detection effectiveness of MT increases as the number of MRs used for testing increases. However, after a certain number of MRs, the rate of increase in fault detection slows due to factors such as the redundancy of MRs. Therefore when there is no significant increase in the fault detection between two MR sets of consecutive sizes of size m and $m + 1$, where the MR set of size $m + 1$ is created by adding one MR to the MR set set of size m, the effective MR set size can be determined. That is, if the difference in fault detection effectiveness of MR set of size m and MR set of size $m+1$ is less than some threshold value, m would be the effective MR set size that should be used for testing. Determining this threshold value should be done considering the critical nature of the SUT. In this work, we used two threshold values of 5% and 2.5% as used in previous related work for determining the oracle data set size [28].

3. We used the following approach to find the *average time taken to detect a fault*: for each killable mutant m in F_v, we computed the time taken to kill the mutant (t_m) by computing the time taken to execute the source and follow-up test cases of the MRs in a given prioritized order until m is killed (here it is assumed that the source and follow-up test cases for each MR are executed sequentially, even though in practice it might be possible to execute them in parallel). Then the average time taken to detect a fault is computed using the following formula:

$$\frac{\sum t_m}{\#\text{ killable mutants in } F_v} \tag{4.10}$$

4. We use a metric called *average percentage of fault detected* (APFD) developed by Elbaum et al. [25] [54] [26] that measures the average rate of fault detection per percentage of test suite execution. The APFD is calculated by taking the

weighted average percentage of faults detected by the MRs. APFD can be calculated using the formula below:

$$\text{APFD} = \frac{1 - MRF_1 + MRF_2 + ... + MRF_m}{nm} + \frac{1}{2n} \tag{4.11}$$

Where MR represents the metamorphic relations under evaluation, m represents the number of faults present in the SUT and n represents the total number of MRs used.

4.4.3 Subject Programs and MRs

We applied the proposed approach on the following three applications from diverse domains for evaluating our proposed MR prioritization methods.

- BBMap[1]: performs global alignment of RNA and DNA sequence reads. The inputs to the program are a set of reads and a reference genome that is used as the alignment reference. The output is the mapping of reads to the reference genome.

- LingPipe[2]: a broad tool kit for processing text using computational linguistics and often used in bioinformatics for bio-entity recognition. In this work, we only use the "bio-entity recognition" functionality that focuses on extracting biomedical terms from text and assigning them to specific categories. Input to the bio-entity recognition problem is a set of bio-medical article and it produces extracted bio-entities as the output.

- Linear Regression[3]: Linear Regression is a linear approach that models the

[1] `https://jgi.doe.gov/data-and-tools/bbtools/bb-tools-user-guide/BBMap-guide/`
[2] `http://alias-i.com/`
[3] `https://weka.sourceforge.io/doc.dev/weka/classifiers/functions/LinearRegression`

relationship between the input variables and the single output variable. The subject program is the linear regression implementation in the Weka library. Weka latest stable version is 3.8.5, which consists of the multiple Linear Regression, named *LinearRegression()* class. The input to the linear regression program is a training data set, and a test data set to represent in .arff format. The output is the prediction made on the instances in the test data set.

- Naive Bayes[4]: A Naive Bayes classifier is a probabilistic machine learning model that's used for the classification task. The subject program is the Naive Bayes algorithm implementation, named *NaiveBayes()* class in the Weka library. The input to the Naive Bayes program is a training data set, and a test data set to represent in .arff format. The output is a classification made on the instances in the test data set.

4.4.4 Metamorphic Relations

To conduct MT on the above-mentioned subject programs, we used a set of MRs developed in previous studies. For testing BBMap, we used eight MRs (listed in Appendix A.1.1) developed by Giannoulatou et al. based on the expected behavior of short-read alignment software [29]. All of these MRs specify modifications to the reads supplied as the input, such as shuffling the reads randomly, duplicating reads, and removing reads.

For conducting MT on IBk, we used 11 MRs developed by Xie et al. [87] (listed in Appendix A.1.2). These MRs are developed based on the user expectations of supervised classifiers. These MRs modify the training and testing data so that the predictions do not change between the source and follow-up test cases.

[4] `https://weka.sourceforge.io/doc.dev/weka/classifiers/bayes/NaiveBayes.html`

For testing the bio-entity recognition task in LingPipe, we used ten MRs that were developed in our previous work [77]. Appendix A.1.4 lists these MRs with additional constraints added to them to make them more generalizable. We describe three categories of MRs for testing bio-entity recognition software. These MR categories are addition, deletion, and shuffling. In addition relations, we extend the text by adding new text such that the sentence/paragraph boundaries are preserved. In the deletion relation, the text is truncated by removing part of it such that the sentence boundaries are preserved. In the shuffling relations, we shuffle portions of the text. Similarly, for testing Linear Regression system, we used 11 MRs developed by Luu et al. [51]. The properties of the linear regression is related to the addition of data points, the rescaling of inputs, the shifting of variables, the reordering of data, and the rotation of independent variables. These MRs were grouped into 6 categories and derived from the properties of the targeted algorithm.

4.4.5 Source and Follow-up Test Cases

As we described before, MT involves the process of generating source and follow-up test cases based on the MRs used for testing. BBMap requires a set of reads and a reference genome as inputs. We used the E.coli reference genome and a set of its reads and Yeast reference genome and a set of its reads [5] as source test cases. E.coli and Yeast are considered model genomes and are widely used in the bioinformatics domain for conducting tests. To generate the follow-up test cases using these source test cases, we applied the input transformations described in the MRs listed in Appendix A.1.1.

IBk uses a training data set to train the k-nearest neighbor classifier, and a test data set is used to evaluate the performance of the trained classifier. Figure A.1 in Appendix A.1.2 shows an example dataset that is used as a source test case for

[5] http://genome.ucsc.edu/

IBk; the training data set is on the left and the test data set is on the right in the figure. After executing this source test case, the output will be the class labels predicted for the test data set, which is a value from the set {0,1,2,3,4} in this example. We generated ten similar datasets randomly, where each training and test data set contained four numerical attributes and a class label. The attribute values are randomly selected within the range [0, 100]. The values of the class label are randomly selected from the set {0,1,2,3,4,5}. The size of the training testing data sets ranges within [0, 200]. To generate the follow-up test cases using these source test cases, we applied the input transformations described in the MRs listed in Section A.1.2.

For creating source test cases for LingPipe, two biomedical articles with the PMCIDs 100320 and 100325 were obtained through PubMed [6]. The sentences and paragraphs used as source test cases for the MRs were picked randomly from these two articles. However, for specific MRs related to removing some words from a list of random words (MR8) and shuffling a list of random words (MR10), 15 random articles were selected, and two sets of 500 random words were chosen from the selected articles to generate the source test case. To generate the follow-up test cases using these source test cases, we applied the input transformations described in the MRs listed in Section A.1.4.

Linear Regression uses a training data set to train the model, and a test data set is used to evaluate the performance of the trained model. We obtained the training data from the machine learning repositories [7],[8] as the source data sets. The first data set contains 515345 instances and 90 attributes. The second data set contains 21263 instances and 81 attributes. To generate the follow-up data sets using these source data sets, we applied the input transformations described in the MRs. Similarly, for

[6]https://www.ncbi.nlm.nih.gov/pubmed
[7]https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD
[8]https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data

Naive Bayes, we obtained the training data from the machine learning repositories [9],[10] as source data sets. The first data set contains 48842 instances and 14 attributes. The second data set contains 400 instances and 5 attributes. To generate the follow-up data set using these source data set, we applied the input transformations described in the MRs listed in Section A.1.3.

4.4.6 Mutant Generation

For each version of the subject program, we aimed at developing generated mutant set to be used as F_v. For this, we used three automated mutation tools: μJava[11], PIT[12] and Major[13]. For generating mutants using μJava, we used all the method level mutation operators [52]. With PIT mutation tool [23] and the Major mutation tool [41] we used all the available mutation operators provided by the tools. All the mutants were generated using these tools were *first order mutants*, where a mutation operator is used to make a single modification to the source code to generate the mutant. Table 4.2 shows the number of mutants used for evaluation.

Table 4.2: Mutants generated for version v_k of the SUTs

Version	**Subject**	**#Major**	**#PIT**	**# μJava**
38.71	BBMap	102	100	N/A
3.8.6	Linear Regression	377	73	N/A
3.8.6	Naive Bayes	452	452	N/A
4.1.0	LingPipe	100	N/A	100

[9]https://archive.ics.uci.edu/ml/datasets/Adult
[10]https://www.kaggle.com/datasets/rakeshrau/social-network-ads
[11]https://github.com/jeffoffutt/muJava
[12]http://pitest.org/
[13]http://mutation-testing.org/

Table 4.3: Validation Setup

Subject	$\mathbf{T_{sp}}$	***SUT Version***	$\mathbf{T_{sv}}$	$\mathbf{F_v}$
BBMap	Ecoli	38.70	Yeast	PIT (38.71)
	Yeast	38.70	Ecoli	Major (38.71)
LingPipe	ID:100325	3.9.3	ID:100320	PIT (4.1.0)
	ID:100320	3.9.3	ID:100325	PIT (4.1.0)
Linear Regression	YearPrediction	3.8.5	SuperConduct	Major (38.71)
	SuperConduct	3.8.6	YearPrediction	PIT (4.1.0)
Naive Bayes	Adult	3.8.5	SocialNetwork	PIT (4.1.0)
	SocialNetwork	3.8.6	Adult	Major (38.71)

4.5 Internal Variable Information

Table 4.4 provides the number of internal variables used for each of the subjects under test. We considered variables which are declared within the body of a method. The internal variables were selected based on the steps described in Section 4.3.1.

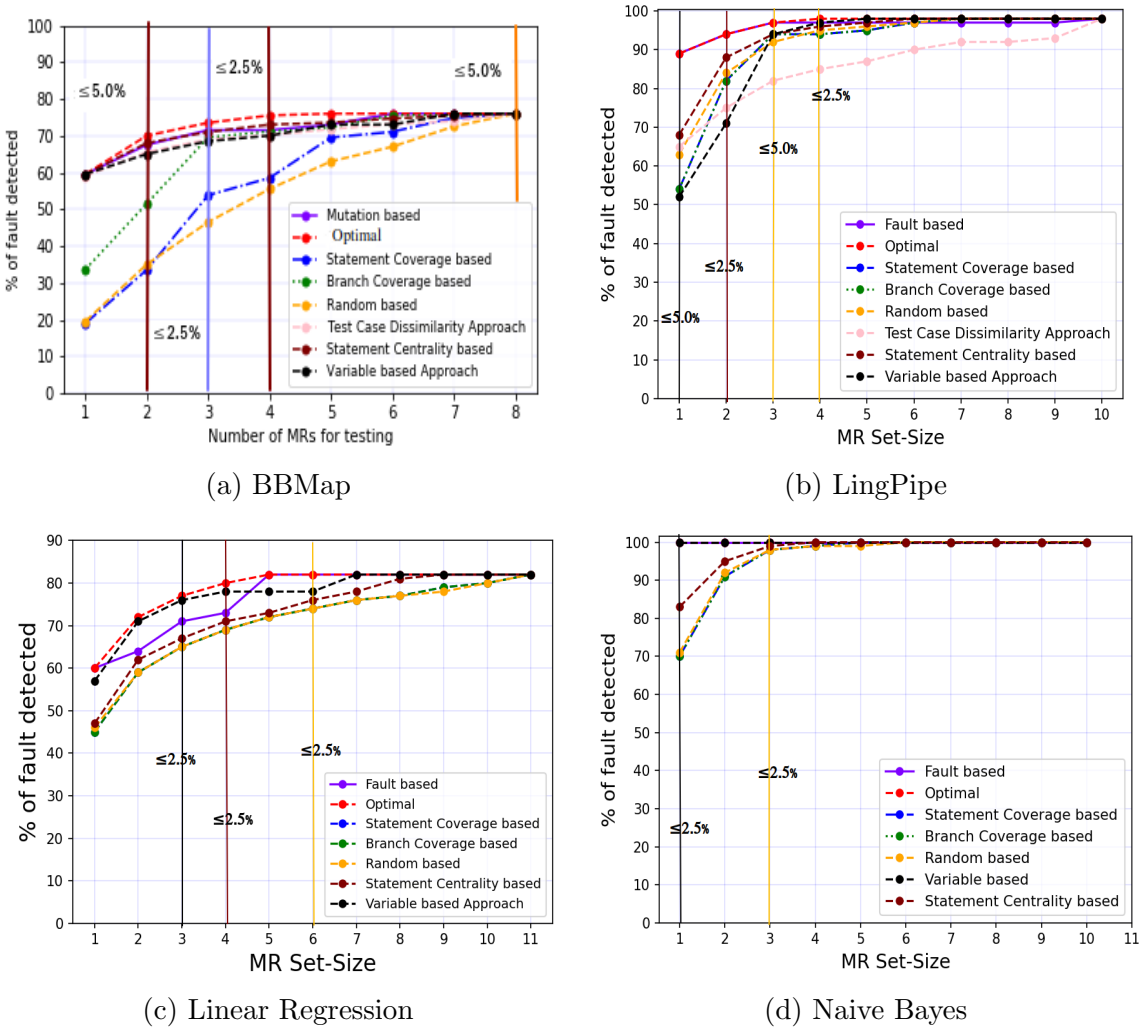
4.6 Results and Discussion

In this Section, we discuss our experimental results and provide answers to the four research questions that we listed in Section 4.4. For each subject program, we carried out the validation procedure described in Section 4.4.1 using the setup described in Table 4.3. In this setup, we used the generated mutant sets and the source test cases as T_{sp}, F_v, and T_{sv} in two different configurations. For example, the

Table 4.4: Internal Variable Information

Subject	**# Internal Variables**
BBMap	28
LingPipe	15
Linear Regression	15
Naive Bayes	10

first evaluation run for BBMap was conducted by executing Ecoli as T_{sp} on version 38.70 (v_{k-1}), Yeast as T_{sv}, and PIT mutants generated on version 38.71 (v_k) as F_v (refers to row 1 of Table 4.3). The second evaluation run for BBMap was conducted by executing Yeast as T_{sp} on version 38.70 (v_{k-1}), Ecoli as T_{sv}, and Major mutants generated on version 38.71 (v_k) as F_v. In Table 4.3 for IBk, Test1 refers to 5 of the randomly generated datasets and Test2 refers to the other five datasets. Figure 4.7 shows the average fault detection effectiveness for evaluation runs described above vs. the MR set size used for testing, for each subject program. We also plot the percentage of faults detected by the random baseline, Fault-based ordering, Code Coverage-based ordering and the Optimal MR ordering for comparison.



(a) BBMap (b) LingPipe

(c) Linear Regression (d) Naive Bayes

Figure 4.7: Fault detection effectiveness for BBMap, LingPipe, Linear Regression and Naive Bayes

4.6.1 RQ1: Comparison of the MR Prioritization approaches against the Random Baseline

4.6.1.1 Fault detection effectiveness We formulated the following statistical hypothesis to answer RQ1 in the context of fault detection effectiveness:

H_1: For a given MR set of size m, the fault detection effectiveness of the MR set produced by Statement Centrality-based approach is higher than that of the

random baseline.

H_2: For a given MR set of size m, the fault detection effectiveness of the MR set produced by Variable-based approach is higher than that of the random baseline.

The null hypothesis H_{0x} for each of the above-defined hypotheses H_X is that the Statement Centrality-based and Variable-based approaches perform equal to or worse than the random baseline.

In Table 4.5 and 4.6, we list the relative improvement in average fault detection effectiveness for the two MR prioritization methods over the currently used random approach for BBMap, LingPipe, Linear Regression, and Naive Bayes respectively. To evaluate the above hypotheses, we used the paired permutation test since it does not make any assumptions about the underlying distribution of the data [31]. We apply the paired permutation test to each MR set size for each of the subject programs with α= 0.05, and the relative improvements that are significant are marked with a asterisk. As shown in Table 4.5 and Table 4.6 for BBMap, the two prioritization approaches showed significant improvements in average fault detection effectiveness over the random approach for all the MR set sizes except for the last set size. Particularly, among the different MR set sizes, the increase in fault detection percentage varies from 4.10% to 215.78% between the two methods. Therefore, we reject the null hypotheses H_{01} and H_{02} for BBMap. Similarly for LingPipe, our proposed approaches shows improvements for MR set sizes $m = 1$ to $m = 6$. Also overall, the relative improvement across MR prioritization methods varies between 0% - 7%. Therefore we reject H_{01} and H_{02} in general for LingPipe. For Linear Regression our proposed approaches show improvement over the random approach for all the MR set sizes. Similarly, for Naive Bayes, our proposed approaches show improvement in fault detection effectiveness for MR set size $m = 1$ to $m = 5$. Therefore, we reject the

null hypotheses H_{01} and H_{02} for Linear Regression and Naive Bayes.

Table 4.5: Relative improvement in fault detection effectiveness of Statement Centrality-based approach compared to random baseline for subject programs

MR Set Size	**BBMap**	**LingPipe**	**Linear Regression**	**Naive Bayes**
1	215.78%*	7.93%*	2.17%*	16.40%*
2	100%*	4.76%*	5.08%*	2.70%*
3	59.09%*	2.17%8	3.07%*	0.81%*
4	35.84%*	1.05%*	2.89%*	1.01%*
5	17.74%*	1.04%*	1.38%	0.41%
6	16.92%*	1.03%*	2.70%*	0%
7	4.10%*	0%	2.63%*	0%
8	0%	0%	5.19%*	0%
9		0%	5.12%*	0%
10		0%	2.5%*	0%
11			0%	

4.6.1.2 Effective number of MRs used for Testing In Figures 4.7a, 4.7b, 4.7c and 4.7d, we show the effective number of MRs using the fault detection thresholds for BBMap, LingPipe, Linear Regression and Naive Bayes respectively. The brown vertical lines represent effective MR set size when using the Statement Centrality-based prioritization, the black vertical line represents the Variable-based and the orange vertical lines represent the same for the random baseline. The respective thresholds are shown near each vertical line.

As shown in Figure 4.7a, for BBMap, the effective MR set size is 2 for the 5% threshold when statement-centrality and Variable-based prioritization is used. With

Table 4.6: Relative improvement in fault detection effectiveness of Variable-based approach compared to random baseline for subject programs

MR Set Size	BBMap	LingPipe	Linear Regression	Naive Bayes
1	215.78%*	-17.31%	23.91%*	40.25%*
2	91.17%*	-15.47%	20.33%*	8.10%*
3	54.54%*	2.17%*	16.92%*	1.83%*
4	32.07%*	2.10%*	13.04%*	1.01%*
5	17.74%*	2.08%*	8.33%*	0.41%*
6	12.30%*	1.03%	5.40%*	0%
7	4.10%*	0%	7.89%*	0%
8	0%	0%	6.49%*	0%
9		0%	5.12%*	0%
10		0%	2.5%*	0%
11			0%	

the random baseline, the effective MR set size is 8. For LingPipe, the effective MR set size is one for the 5.0% threshold with Variable-based prioritization and MR set size is two for the 2.5% threshold when a Statement Centrality-based approach is used. With the random baseline, 5.0% thresholds are only met with MR set size 3 and the 2.5% threshold is met with MR set size 6. Similarly for Linear Regression, MR set size is 3 for the 2.5% when the Variable-based approach is used and MR set size is 4 for the 2.5% threshold when the Statement Centrality-based approach is used. In comparison with random baseline and Code Coverage-based, MR set size is 6 for the 2.5% threshold. For Naive Bayes, MR set size is 1 is met at 2.5% threshold when Variable-based is used and MR set size is 3 is met at 2.5% when Statement Centrality-based is used. With a random baseline, MR set size is 3 for a 2.5% threshold.

Therefore for all the four subject programs, using Statement Centrality and Variable-based prioritization resulted in a reduction in the effective MR set size compared to using a random and Code Coverage-based ordering of MRs.

4.6.1.3 The average time taken to detect a fault Tables 4.7 and 4.8 shows the average time taken to detect a fault in BBMap and LingPipe respectively, for the two validation configurations. As shown in these results, using Statement Centrality-based prioritization resulted in significant reductions in the average time taken to detect a fault ranging from 58%-79% relative to the random baseline for BBMap and 26%-47% relative to the random baseline for LingPipe.

4.6.1.4 Average percentage of fault detected Table 4.9 shows the APFD for the SUT. We can observe that for all our subject programs, our proposed approach provided a higher APFD value between 0.87-0.97 and shows an average 15% increase in rate of fault detection when compared to random-based prioritization. As a result, our approach provides a higher rate of fault detection when compared to the random approach.

Table 4.7: Comparison of average time taken to detect a fault using Fault-based MR prioritization and Random baseline for BBMap

T_{sv}	**F_v**	**Statement Centrality-based**	**Random baseline**	**% Time reduction**
Yeast	PIT	85.23s	402.09s	79%
Ecoli	Major	55.22s	168.21s	58%

Table 4.8: Comparison of average time taken to detect a fault using Statement Centrality-based MR prioritization and Random baseline for LingPipe

$\mathbf{T_{sv}}$	$\mathbf{F_v}$	**Statement Centrality-based**	**Random baseline**	**% Time reduction**
ID:100325	PIT	96s	130s	26.15%
ID:100320	Major	120s	230s	47.82%

Table 4.9: Average percentage of fault detected for subject programs

Subject	**Statement Centrality Based**	**Variable Based**	**Random Based**
BBMap	0.91	0.93	0.65
LingPipe	0.87	0.90	0.76
Linear Regression	0.92	0.97	0.83
Naive Bayes	0.90	0.95	0.88

4.6.2 RQ2: Comparison of proposed MR Prioritization approaches against Optimal Ranking

In order to answer the second research question, we formulated the following statistical hypothesis:

H_3: For a given MR set of size m, the fault detection effectiveness of the MR set produced using Optimal ordering is equal to or higher than the fault detection effectiveness of the MR set produced by the Statement Centrality prioritization.

Table 4.10: Average relative improvement in fault detection effectiveness of Optimal approach over Statement Centrality-based approach for SUT

MR Set-Size	**BBMap**	**LingPipe**	**Linear Regression**	**Naive Bayes**
1	0%	30.88%*	0%	20.48%*
2	0%	6.81%*	0%	5.26%*
3	1.42%*	3.19%*	1.42%*	1.01%*
4	0%	2.08%*	0%	0%
5	0%	1.03%*	0%	0%
6	0%	0%	0%	0%
7	0%	0%	0%	0%
8	0%	0%	0%	0%
9		0%	0%	0%
10		0%	0%	0%
11			0%	

H_4: For a given MR set of size m, the fault detection effectiveness of the MR set produced using Optimal ordering is equal to or higher than the fault detection effectiveness of the MR set produced by Variable-based prioritization.

The null hypothesis H_{0x} for each of the above-defined Hypotheses H_x is that the MR sets generated by the corresponding MR prioritization approaches perform worse than the MR sets generated by the Optimal ordering in terms of fault detection effectiveness. Table 4.10 and 4.11 show the relative improvement in fault detection effectiveness between the Optimal ordering and our proposed MR prioritization methods. The results that are significant at $\alpha = 0.05$ are shown with a $*$. As discussed above, Optimal ordering represents the upper bound of fault detection for

Table 4.11: Average relative improvement in fault detection effectiveness of Optimal approach over Variable-based approach for SUT

MR Set-Size	**BBMap**	**LingPipe**	**Linear Regression**	**Naive Bayes**
1	2.17%*	71.15%*	5.26%*	0%
2	13.72%*	32.39%*	1.40%*	0%
3	12.5%*	3.19%*	1.31%*	0%
4	3.22%*	1.30%*	2.56%*	0%
5	3.22%*	0	5.12%*	0%
6	1.58%*	0	5.12%*	0%
7	1.58%*	0	0	0%
8	0%	0	0	0%
9		0	0	0%
10		0	0	0%
11			0	

a given set of MRs. Here we attempt to find how closely our MR prioritization approach performs compared to this upper bound.

As shown in Table 4.10, for BBMap, the Statement Centrality-based approach performed equal to the Optimal based ordering for all the MR set sizes except for MR set size $m = 3$. Similarly for Linear Regression, the Statement Centrality-based approach performs equal to Optimal ordering for all the MR set sizes. Therefore, we reject hypothesis H_3 for BBMap and Linear Regression. For the Naive Bayes program, the Statement Centrality approach performs equal to Optimal ordering for MR set size $m = 4$ to $m = 10$ and reported a relative improvement in fault detection effectiveness of less than 5% when compared to Optimal ordering for MR set size $m = 2$ and $m = 3$. Therefore, we cannot reject hypothesis H_3 for Naive Bayes. Similarly, for LingPipe,

the Statement Centrality-based approach performs equal to Optimal ordering from MR set size $m = 6$ to $m = 10$ and relative improvement is less than 10% when compared to Optimal ordering for MR set size $m = 3$ and $m = 4$. Therefore, a Statement Centrality-based approach performed closer to Optimal ordering for all the subject programs.

Table 4.11 shows the relative improvement in fault detection effectiveness between Variable-based approach and Optimal ordering. Variable-based approach performs equal to Optimal ordering for all the MR set sizes for the Naive Bayes program. Therefore, we can reject the hypothesis H_4 for Naive Bayes. For Linear Regression, the Variable-based approach reported a relative improvement in fault detection effectiveness of less than 5% when compared to Optimal ordering for all MR set sizes. Similarly for LingPipe, MR set size $m = 5$ to $m = 10$ performs equal to Optimal ordering and also reported relative improvement of less than 5% in fault detection when compared to Optimal ordering for MR set size $m = 3$ and $m = 4$. Therefore, we cannot reject the H_4 for Linear Regression and LingPipe. However, a Variable-based approach performed closer to Optimal ordering for all the subject programs.

In general, the relative improvement in fault detection effectiveness of the proposed method is small in comparison to Optimal ordering. Therefore, Statement Centrality and Variable-based MR prioritization are effective in producing an MR ordering that is comparably effective as the Optimal ordering.

4.6.3 RQ3: Comparison of Statement Centrality-based and Variable-based MR prioritization approach against the coverage-based prioritization approaches

In this research question, we evaluate whether the Statement Centrality-based and Variable-based ranking approaches outperform statement and branch coverage-

Table 4.12: Average relative improvement in fault detection effectiveness using statement-centrality based approach over Statement coverage-based prioritization approaches

MR Set-Size	**BBMap**	**LingPipe**	**Linear Regression**	**Naive Bayes**
1	31.33%*	26.18%*	4.44%	19.08%*
2	30.76%*	14.67%*	5.08%*	3.93%*
3	7.69%*	6.27%*	3.07%*	1.02%
4	7.46%*	6.08%*	2.89%	0.90%
5	5.79%*	4.61%*	1.38%	0.30%
6	8.57%*	2.92%*	2.70%*	0%
7	5.55%*	2.67%*	2.63%*	0%
8	2.74%	2.74%	5.19%*	0%
9		0	3.79%*	0%
10		0	2.5%	0%
11			0	

based approaches. To answer the research question, we formulate the following hypotheses:

H_5: For a given MR set of size m, the fault detection effectiveness of the MR set produced using Statement Centrality-based prioritization is higher than the fault detection effectiveness of the MR set produced by the statement coverage-based and branch coverage-based prioritization.

H_6: For a given MR set of size m, the fault detection effectiveness of the MR set produced using Variable-based prioritization is higher than the fault detection effectiveness of the MR set produced by the statement coverage-based and

Table 4.13: Average relative improvement in fault detection effectiveness using statement-centrality based approach over branch coverage-based prioritization approaches

MR Set-Size	**BBMap**	**LingPipe**	**Linear Regression**	**Naive Bayes**
1	13.33%*	26.18%*	4.44%*	19.08%*
2	21.42%*	14.67%*	5.08%*	3.93%*
3	7.69%*	6.27%*	3.07%*	1.02%*
4	7.46%*	6.08%*	2.89%*	0.90%*
5	5.79%*	4.61%*	1.38%*	0.30%*
6	8.57%*	2.92%*	2.70%*	0%
7	5.55%*	2.67%*	2.63%*	0%
8	0%	2.74%*	5.19%*	0%
9		0%	3.79%*	0%
10		0	2.5%*	0%
11			0%	

branch coverage-based prioritization.

The null hypothesis H_{0x} for each of the above-defined hypotheses H_x is that the MR sets generated by the Statement Centrality-based and Variable-based prioritization method perform equal to or worse than the MR sets generated by the statement and branch coverage-based prioritization in terms of fault detection effectiveness.

Table 4.12 and Table 4.13 show the relative improvement in fault detection percentage between the MR sets generated by the Statement Centrality-based prioritization approach and the MR sets generated by a statement coverage and branch-based prioritization approaches for BBMap, LingPipe, Linear Regression,

Table 4.14: Average relative improvement in fault detection effectiveness using Variable-based approach over statement and branch coverage-based prioritization approaches

MR Set-Size	**BBMap**	**LingPipe**	**Linear Regression**	**Naive Bayes**
1	33.33%*	-3.70%	26.66%*	43.37%*
2	25%*	-13.41%*	20.03%*	9.40%*
3	4.61%*	0%	16.92%*	2.04%*
4	4.47%*	3.91%*	13.04%*	0.90%
5	5.79%*	3.15%*	8.33%*	0.30%
6	8.57%*	1.03%	5.40%*	0%
7	5.55%*	0%	7.89%*	0
8	0%	0%	6.49%8	0
9		0%	3.79%*	0%
10		0	2.5%*	0%
11			0%	

and Naive Bayes respectively. For the BBMap program, Statement Centrality-based prioritization had a higher relative improvement in fault detection effectiveness compared to the statement and branch coverage-based prioritization for all the MR set sizes. Therefore, we can reject the null hypothesis H_5 for BBMap. Similarly, for the Linear Regression program, the Statement Centrality-based approach had higher fault detection effectiveness compared to the statement and branch coverage-based approach for all the MR set sizes. Therefore, we can reject the null hypothesis H_5 for Linear Regression. For Naive Bayes, the Statement Centrality-based approach provided higher fault detection effectiveness for MR set size $m = 1$ to $m = 5$ when compared to statement and branch coverage-based prioritization. Therefore,

in general, we can reject the null hypotheses H_{05} for Naive Bayes only for MR set sizes $m = 1$ to $m = 5$.

Table 4.14 shows the relative improvement in fault detection percentage shown by the Variable-based prioritization approach and Code Coverage-based prioritization approaches for BBMap, LingPipe, Linear Regression, and Naive Bayes respectively. For BBMap and Linear Regression programs, Variable-based prioritization provided higher relative improvement in fault detection effectiveness compared to statement and branch coverage-based prioritization for all MR set sizes. Therefore, we can reject the null hypotheses H_{06} for BBMap and Linear Regression. For Naive Bayes, the Statement Centrality-based approach provided higher relative improvement in fault detect effectiveness for MR set size $m = 1$ to $m = 5$ when compared to branch coverage. Therefore, in general, we can reject the null hypotheses H_{06} for Naive Bayes only for MR set size $m = 1$ to $m = 5$.

In practice, these increases in improvements in fault detection effectiveness translate to potentially detecting more faults using our proposed approach when compared to coverage-based MR prioritization methods.

4.6.4 RQ4: Comparison of Statement Centrality-based and Variable-based MR prioritization approach against the Fault-based prioritization approach

In this research question, we evaluate whether the Statement Centrality-based and Variable-based ranking approaches outperform Fault-based approaches. To answer the research question, we formulate the following hypotheses:

H_7: For a given MR set of size m, the fault detection effectiveness of the MR set produced using Statement Centrality-based prioritization is higher than the fault detection effectiveness of the MR set produced by the Fault-based prioritization.

H_8: For a given MR set of size m, the fault detection effectiveness of the MR set produced using Variable-based prioritization is higher than the fault detection effectiveness of the MR set produced by the Fault-based prioritization.

The null hypothesis H_{0x} for each of the above-defined Hypotheses H_x is that the MR sets generated by the Statement Centrality-based and Variable-based prioritization method perform equal to or worse than the MR sets generated by the Fault-based prioritization in terms of fault detection effectiveness.

Table 4.15 shows the relative improvement in fault detection percentage between the MR sets generated by the Statement Centrality-based prioritization approach and the MR sets generated by Fault-based prioritization approaches for BBMap, LingPipe, Linear Regression, and Naive Bayes respectively. For the BBMap program, the Statement Centrality-based approach selected MR with the highest fault detection effectiveness and placed it first in the prioritized MR order. Subsequently, from MR set sizes 2 to 8, the proposed approach showed higher relative improvement in the fault detection when compared to the Fault-based. Therefore we can reject the null hypotheses H_{07} for BBMap. For the LingPipe program, the Fault-based approach provides less than 5% improvement in fault detection effectiveness for MR set sizes $m = 2$ to $m = 10$ when compared to the Statement Centrality-based approach. Therefore, we can not reject the null hypothesis H_{07} for LingPipe. Similarly for Linear Regression, the Fault-based approach provided a relative improvement in fault detection of less than 10% when compared to the Statement Centrality approach for MR set sizes $m = 1$ to $m = 8$. Therefore, we can reject the null hypothesis H_{07} for Linear Regression only for MR set sizes $m = 1$ to $m = 8$. For Naive Bayes, the Statement Centrality-based approach performs equal to the Fault-based approach for MR set size $m = 4$ to $m = 10$. Therefore, we can reject the null hypotheses H_{07} for the Naive Bayes program. Also, the Statement Centrality approach performed closer

to the Fault-based approach for all the subject programs.

Table 4.16 shows the relative improvement in fault detection percentage between the MR sets generated by the Variable-based prioritization approach and the MR sets generated by Fault-based prioritization approaches for BBMap, LingPipe, Linear Regression, and Naive Bayes respectively. For the BBMap program, the Variable-based approach showed higher relative improvement in the fault detection when compared to the Fault-based for all the MR set sizes $m = 3$ to $m = 8$. Therefore we can reject the null hypotheses H_{08} for BBMap. For the LingPipe program, the Fault-based approach provides less than 15% improvement in fault detection effectiveness for MR set sizes $m = 2$ to $m = 10$ when compared to the Variable-based approach. Therefore, we cannot reject the null hypothesis H_{08} for LingPipe. Similarly for Linear Regression, the Variable-based approach provided higher fault detection from MR set sizes $m = 2$ to $m = 4$ when compared to the Fault-based. Also, Variable-based approach performed is equal to Fault-based approach for MR set size $m = 6$ to $m = 11$. Therefore, we cannot reject the null hypothesis H_{08} for Linear Regression. Similarly, for Naive Bayes, the Statement Centrality-based approach performs equal to the Fault-based approach for all the MR set sizes. Therefore, we cannot reject the null hypotheses H_{08} for Naive Bayes and Variable-based approach performed closer to the Fault-based approach for the subject programs. Our results indicated that the proposed approaches outperformed Fault-based approach for BBMap. For Lingpipe, Linear Regression and Naive Bayes, Fault-based approach provided 1-10% increase over our proposed approaches and thus performed closer to Fault-based approach.

4.7 Threat to Validity

External Validity: Statement Centrality-based approach does not work effectively for programs having limited use, definition, and dependencies of data

Table 4.15: Average relative improvement in fault detection effectiveness using Fault-based approach over Statement Centrality-based prioritization approach

MR Set-Size	**BBMap**	**LingPipe**	**Linear Regression**	**Naive Bayes**
1	0%	16.50%*	27.65%*	20.48%*
2	-4.41%*	3.06%*	3.22%*	5.26%
3	-4.28%*	2.81%*	5.97%*	1.01%
4	-6.94%*	2.28%*	2.81%*	0%
5	-4.10%*	1.19%*	12.32%*	0%
6	-5.26%*	0.94%*	7.89%*	0%
7	-3.94%*	0.67%	5.12%*	0%
8	0%	0.48%	1.23%*	0%
9		0.05%	0%	0%
10		0%	0%	0%
11			0%	

in programs. This issue is mitigated using the data diversity-based approach for prioritization of MRs discussed in Chapter 5. We have generated approximately 200 mutants for the validation of our proposed approach for each subject. However, the number of mutants used may be low.

Internal Validity: We selected only local variables and array variables were excluded to reduce the dynamic analysis and source code monitoring time. Also, the number of variables was selected and determined considering the execution time of the MRs. However, the number of internal variables selected could be low. In a situation when the coverage of source and follow-up test case for an MR is the same, then the Statement Centrality-based approach becomes ineffective. However, in such scenarios, Variable-based approach could be a better alternative to use.

Table 4.16: Average relative improvement in fault detection effectiveness using Fault-based approach over Variable-based prioritization approach

MR Set-Size	**BBMap**	**LingPipe**	**Linear Regression**	**Naive Bayes**
1	0%	34.60%*	5.26%*	0%
2	0%	18.13%*	-9.85%*	0%
3	-1.47%*	12.72%*	-6.57%*	0%
4	-4.28%*	10.38%*	-6.41%*	0%
5	-4.10%*	10.30%*	5.12%*	0%
6	-1.36%	9.89%*	5.12%*	0%
7	-3.94%*	6.93%*	0%	0%
8	0%	4.45%*	0%	0%
9		1.07%	0%	0%
10		0%	0%	0%
11			0%	

4.8 Summary

In this work, we developed automated approaches for MR prioritization to improve the efficiency and effectiveness of MT. These approaches use (1) Statement Centrality information and (2) internal variable values to prioritize MRs. We conducted an empirical study to evaluate the effectiveness of the developed approaches using four complex open-source software systems and machine learning programs. Our results indicate that the Statement Centrality and Variable-based approach outperforms the random and Code Coverage-based approach for subject programs. Also, our proposed approaches performed closer to Fault-based and Optimal ordering for the subject programs. Our results indicated that the proposed approach reduced

the time taken to detect a fault by 79% for BBMap and 47% for LingPipe. Also, the average percentage of fault detected by our proposed approaches was higher than the random approach.

CHAPTER FIVE

DATASET DIVERSITY APPROACH

5.1 Motivation

Machine learning (ML) is increasingly deployed in large-scale software and safety-critical systems due to recent advancements in deep learning and reinforcement learning. The software applications powered by ML are applied and utilized in our daily lives; from finance, and energy, to health and transportation. AI-related accidents are already making headlines, from inaccurate facial recognition systems causing false arrests to unexpected racial and gender discrimination by machine learning software [46] [96] [30]. Also, a recent incident with an Uber autonomous car resulted in the death of a pedestrian [63]. Thus, ensuring the reliability and conducting rigorous testing of machine learning systems is very vital to prevent any future disasters.

ML-based systems exhibit non-deterministic behavior and reason in a probabilistic manner. Also, the output is learned and predicted by a machine learning model as result do not have an expected output to compare against the actual output. As a result, the correctness of the output produced by machine learning-based applications cannot be easily determined and suffer from test oracle problem [47, 56, 60, 91].

Previous work [77] has shown that several MRs detect the same type of fault and differ only in fault detection effectiveness. Also, each MR in a machine learning application can have multiple source and follow-up test cases, as a result, the execution time of the MRs increases drastically. The source and follow-up test cases in an MR represented as training data set could take several days to train and build a machine learning model for a large training data set. Hence, prioritization of MRs is important

for conducting effective metamorphic testing on these programs. Also, Prior work [76] introduced approaches to prioritize MRs based on Code Coverage and Fault-based approach. Further, the Code Coverage-based approach does not work effectively for machine learning-based applications since the data used for training determines the program logic. A machine learning program is usually just a sequence of library function invocations and as a result, 100% code coverage can be easily achieved when at least one test input is processed [32, 69].

To this end, in this work, we propose methods for *MR prioritization* for machine learning programs. The MR prioritization method proposed in this work qualitatively identifies the diversity between the source and follow-up training data set in an MR. Therefore, we propose four metrics to prioritize metamorphic relations based on diversity in the data set between the source and follow-up test case. Our intuition behind the metrics is that greater the diversity in the data set, then greater the possibility of introducing fault in the machine learning model. Figure 5.1 explains our proposed approach. In the following subsection, I shall explain each of the metrics in detail.

5.2 Proposed Approach

In this Section, we discuss our proposed method for the prioritization of MRs. Figure 5.1 shows the steps for prioritizing MRs. In this approach, we prioritize the MRs as follows:

1. Let the set of source test cases in an MR used for testing the SUT be the *prioritizing source test cases* (T_{sp}).

2. Let the set of follow-up test cases in an MR used for testing the SUT be the *prioritizing follow-up test cases* (T_{fp}).

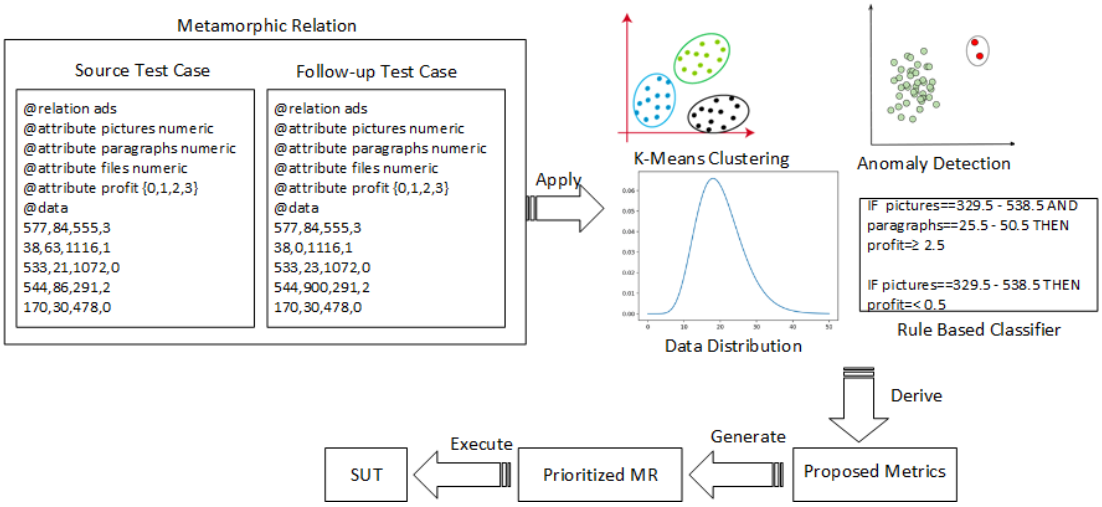


Figure 5.1: Steps for MR prioritization of ML Programs

3. For (T_{sp}) and (T_{fp}), apply the proposed metric: rule based, anomaly detection, Clustering-based and Data Distribution. The metrics are discussed in detail in Section 5.2.1 to Section 5.2.4.

4. For (T_{sp}) and (T_{fp}), apply the proposed metric: rule based, anomaly detection, Clustering-based and Data Distribution and get the MR diversity value of each MR based on the formulas provided in Section 5.2.1 to Section 5.2.4.

5. MR diversity value calculated using the metrics for each MR is normalized using the following steps:

   (a) Let the set of MR diversity value of MRs be DMR_v.

   (b) Identify minimum and maximum value in DMR_v. Let minimum value be Min_v and maximum value be Max_v.

   (c) For each MR diversity value $v \in DMR_v$, generate the normalized MR

diversity value using the formula below:

$$N_{MR} = \frac{v - Min_v}{Max_v - Min_v} \tag{5.1}$$

Where N_{MR} represents the normalized MR diversity value.

6. Prioritize the normalized MR diversity values and select the top n MRs from the prioritized ordering to execute based on the resources available for testing.

Our proposed metrics are explained in detail below.

5.2.1 Rule Based Classifier

Rule-based classifiers are just another type of classifier which makes the class decision by using various if-then rules. We apply a Rule-based classifier to the source and follow-up test cases in a metamorphic relation to determine the diversity between the test cases. We hypothesize that if greater the diversity in the classification rules between the source and follow-up test case, then greater the performance of the model and greater the possibility of detecting a fault in the ML system. We apply the following steps.

1. Let the set of source test cases used for testing SUT be the *prioritizing source test cases* (T_{sp}).

2. Let the set of follow-up test cases of the SUT be the *prioritizing follow-up test cases* (T_{fp}).

3. Apply Sequential Covering Algorithm (CN2) on T_{sp} and T_{fp} and obtain the rules.

4. Remove same rules detected between T_{sp} and T_{fp}.

5. Calculate the total rules identified in T_{sp} and T_{fp}. Let $Total_{sr}$ and $Total_{fr}$ represent the total rules in T_{sp} and T_{fp} respectively.

6. We calculate the diversity of MR using the formula below:

$$R_{MR} = \mid (Total_{sr}) - (Total_{fr}) \mid \tag{5.2}$$

Where R_{MR} represent the diversity of the MR.

5.2.2 Anomaly Detection

Anomaly detection (or outlier detection) is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data. We apply anomaly detection to identify outliers in the source and followup test cases in the metamorphic relation. We hypothesize that greater the diversity in outliers detected between the source and followup, greater the possibility of detecting fault in the machine learning model. We apply the following steps.

1. Let the set of source test cases used for testing SUT be the *prioritizing source test cases* (T_{sp}).

2. Let the set of follow-up test cases of the SUT be the *prioritizing follow-up test cases* (T_{fp}).

3. Apply K Nearest Neighbour (KNN) algorithm to T_{sp} and T_{fp} and identify outlier instances.

4. Remove same outliers detected between T_{sp} and T_{fp}.

5. Calculate the total outliers in T_{sp} and T_{fp} respectively. Let $Total_{so}$ and $Total_{fo}$ be the sum of outlier detected in T_{sp} and T_{fp} respectively.

6. We calculate the diversity of MR using the formula below:

$$O_{MR} = \mid (Total_{so}) - (Total_{fo}) \mid \tag{5.3}$$

Where O_{MR} represent the diversity of the MR.

5.2.3 Clustering-based

In this metric, we partition the input space ie. all possible inputs into different region and determine how the data is distributed in the space. To achieve this, we use clustering methods which simply try to group similar patterns into clusters whose members are more similar to each other based on some distance measure than to members of other clusters. We hypothesize that greater the diversity in patterns detected between the source and followup, greater the possibility of detecting fault in the machine learning model. We apply the following steps.

1. Let the set of source test cases used for testing SUT be the *prioritizing source test cases* (T_{sp}).

2. Let the set of follow-up test cases of the SUT be the *prioritizing follow-up test cases* (T_{fp}).

3. Apply K mean algorithm to T_{sp} and T_{fp} and identify clusters. Let clusters of T_{sp} and T_{fp} be S_{cl} and F_{cl}.

4. Find the distance between the clusters by calculating the euclidean distances between the clusters in S_{cl}. Let the total distance between the clusters be Ts_{cds}.

5. Determine the size of each cluster in S_{cl}. Let the total size of the clusters in S_{cl} be Ts_{sc}.

6. Find the distance within the cluster by calculating the average of the distances from observations to the centroid of each cluster in S_{cl}. Let average distance within the cluster in S_{cl} be As_{cd}.

7. Apply the steps 4 to 6 to F_{cl} respectively. Let Tf_{cds} be the total distance between the clusters, Tf_{sc} be the total size of the clusters and Af_{cd} be the average distance within the cluster.

8. We calculate the diversity of MR using the formula below:

$$C_{MR} = | \ (\mathrm{Ts}_{cds}+Ts_{sc}+As_{cd})\text{-}(Tf_{cds}+Tf_{sc}+Af_{cd}) \ | \tag{5.4}$$

Where C_{MR} is the diversity of MR.

5.2.4 Data Distribution

Data distribution is a function which specifies all the possible values of the data. In this metric, we used shape and spread of Data Distribution of dataset to find diversity of metamorphic relations. The spread of distribution involves calculating range, variance and standard deviation of the source and follow-up test cases. Similarly, shape of distribution involves calculating skewness and kurtosis of the source and follow-up test cases of the MR. We hypothesize that larger the diversity in the data distribution between the source and follow-up test cases of the MR, then greater the possibility of introducing fault in the model. We apply the following steps.

1. Let the set of source test cases used for testing SUT be the *prioritizing source test cases* (T_{sp}).

2. Let the set of follow-up test cases of the SUT be the *prioritizing follow-up test cases* (T_{fp}).

3. Calculate the skewness and kurtosis for T_{sp}. Let ST_{sk} be the sum of skewness and kurtosis for T_{sp}.

4. Calculate range, variance and standard deviation for T_{sp}. Let ST_{rvs} be the sum of variance, range and standard deviation for T_{sp}.

5. Apply step 3 and 4 to T_{fp}. Let FT_{sk} and FT_{rvs} be the shape and spread of distribution for T_{fp}.

6. We calculate the diversity of MR using the formula below:

$$DD_{MR} = |\ (\mathrm{ST}_{sk}+\mathrm{ST}_{rvs})\text{-}(FT_{sk}+FT_{rvs}| \tag{5.5}$$

Where DD_{MR} is the diversity of MR.

5.3 Evaluation

In this work, we plan to evaluate the utility of the developed prioritization approaches on the following aspects: (1) fault detection effectiveness of MR sets, (2) effective number of MRs required for testing, (3) time taken to detect a fault and (4) average percentage of fault detected. We evaluate the effectiveness of the proposed approaches with (1) a *random baseline*: this represents the current practice of executing source and follow-up data set of the MRs in random order. (2) *Code Coverage-based ordering*: this represents the MR order based on the statement and branch-coverage information of the MRs. The approach selects MR which has covered the highest number of statements or branches and place it in the prioritized MR ordering. The process continues until all the possible statements or branches are covered [76]. (3) *Fault-based ordering:* this represents the MR ordering based on mutant kill/alive information for each of the MR which is used to prioritize the

MRs [76]. (4) *Optimal ordering:* this represents the MR order that yields the maximum achievable fault detection effectiveness with the given set of MRs and source/follow-up data set [76].

We conducted experiments to find answers to the following research questions:

1. Research Question 1 (RQ1): *How does the proposed MR prioritization approach based on Data Diversity more effective than the random baseline?*

2. Research Question 1 (RQ2): *How does the proposed MR prioritization approach more effective than the Code Coverage-based prioritization?*

3. Research Question 3 (RQ3): *How does the data-diversity based approach perform against Fault-based approach?*

4. Research Question 4 (RQ4): *How does the data-diversity based approach perform against Optimal ordering of MR?*

5.3.1 Evaluation Procedure

To answer the above research questions, we carry out the following validation procedure similar to previous work [76]:

1. We generate a set of mutants for the SUT. These mutants will be referred to as the *validation set of faults* (F_v). Then from the generated mutants, we remove mutants that give exceptions or mutants that do not finish execution within a reasonable amount of time from further consideration.

2. We use the method described in Section 5.2 to obtain Data Diversity-based MR ordering. Then we applied the obtained MR ordering to F_v and log the mutant killing information.

3. *Creating the random baseline*: We generate 100 random MR orderings and apply each of those orderings to F_v. The mutant killing information for each of those random orderings is logged. The average mutant killing rates of these 100 random orderings were computed to get the fault detection effectiveness of the random baseline.

4. *Creating the Optimal ordering*: Generate by applying the greedy approach to F_v as follows [76]:

   (a) Select the MR that kills the largest number of mutants in F_v and place it in the Optimal ordering.

   (b) Remove the mutants killed by that MR from F_v.

   (c) Repeat the previous two steps until all the possible mutants are killed.

5. *Creating the Fault-based ordering*: Apply greedy approach to F_v as follows [76]:

   (a) Select the MR that detected the highest number of faults in F_v and place it in the prioritized MR ordering.

   (b) Remove each $f \in F_v$ detected as faulty by that MR.

   (c) Repeat steps 5a and 5b until all the possible faults are revealed.

6. *Creating the Code Coverage-based ordering*: Apply greedy approach on the statements or branches covered to prioritize the MRs as follows [76]:

   (a) Select the MR that covers the highest number of statements or branches in the program and place it in the prioritized MR ordering.

   (b) Remove the statements and branches covered by the MR.

   (c) Repeat the previous two steps until all the statements and branches are covered.

5.3.2 Evaluation Measures

We used the following measures to evaluate the effectiveness of the MR orderings generated by our proposed metrics:

1. To measure the *fault detection effectiveness* of a given set of MRs, we use the percentage of mutants killed by those MRs.

2. To calculate the *effective MR set size* we used the following approach: typically, the fault detection effectiveness of MT increases as the number of MRs used for testing increases. However, after a certain number of MRs, the rate of increase in fault detection slows due to factors such as the redundancy of MRs. Therefore when there is no significant increase in the fault detection between two MR sets of consecutive sizes of size m and $m + 1$, where the MR set of size $m + 1$ is created by adding one MR to the MR set of size m, the effective MR set size can be determined. That is, if the difference in fault detection effectiveness of MR set of size m and MR set of size $m+1$ is less than some threshold value, m would be the effective MR set size that should be used for testing. Determining this threshold value should be done considering the critical nature of the SUT. In this work, we used two threshold values of 5% and 2.5% as used in previous related work for determining the oracle data set size [28].

3. We used the following approach to find the *average time taken to detect a fault*: for each killable mutant m in F_v, we computed the time taken to kill the mutant (t_m) by computing the time taken to execute the source and follow-up test cases of the MRs in a given prioritized order until m is killed (here it is assumed that the source and follow-up test cases for each MR are executed sequentially). Then the average time taken to detect a fault is computed using the following

formula:

$$\frac{\sum t_m}{\#killable\,mutants\,in\,F_v}$$

4. We calculate *Average percentage of fault detected* (APFD) developed by Elbaum et al. [25] [54] [26] that measures the average rate of fault detection per percentage of test suite execution. The APFD is calculated by taking the weighted average of the number of faults detected during the execution of MRs. APFD can be calculated using the formula below:

$$\text{APFD} = \frac{1 - MRF_1 + MRF_2 + ... + MRF_m}{nm} + \frac{1}{2n} \tag{5.6}$$

Where MRF_i represents the fault F_i detected by the metamorphic relations under evaluation, m represents the number of faults present in the SUT and n represents the total number of MRs used.

5.3.3 Subject Programs and MRs

We applied the above-mentioned validation procedure to the following four machine learning programs for evaluating our proposed MR prioritization methods.

- IBk[1]: K-nearest neighbours (KNN) classifier implementation in the Weka machine learning library [2]. The input to IBk is a training data set, and a test data set to represent in .arff format. The output is the classification predictions made on the instances in the test data set.

- Linear Regression[2]: linear regression is a linear approach that models the relationship between the input variables and the single output variable. The subject program is the linear regression implementation in the weka library.

[1] `http://weka.sourceforge.net/doc.dev/weka/classifiers/lazy/IBk.html`
[2] `https://weka.sourceforge.io/doc.dev/weka/classifiers/functions/LinearRegression`

Weka latest stable version is 3.8.5, which consists of the multiple linear regression, named *LinearRegression()* class. The input to the linear regression program is a training data set, and a test data set to represent in .arff format. The output is the prediction made on the instances in the test data set.

- Naive Bayes[3]: A Naive Bayes classifier is a probabilistic machine learning model that's used for the classification task. The subject program is the Naive Bayes algorithm implementation, named *NaiveBayes()* class in the Weka library. The input to the Naive Bayes program is a training data set, and a test data set to represent in .arff format. The output is a classification made on the instances in the test data set.

- Convolutional Neural Network[4]: Convolutional Neural Network (CNN) is a class of Artificial Neural Network and most commonly applied to analyze visual imagery. The subject program is the CNN algorithm implementation in Deeplearning4j library. The input to the CNN is MNIST handwritten digit data set and the output is to classify the image of a handwritten digit into one of 10 classes representing integer values from 0 to 9.

5.3.4 Metamorphic Relations

For conducting MT on IBk and Naive Bayes, we used 11 MRs developed by Xie et al. [87]. These MRs are developed based on the user expectations of supervised classifiers. These MRs modify the training and testing data so that the predictions do not change between the source and follow-up test cases. Similarly, for testing the Linear Regression system, we used 11 MRs developed by Luu et al. [51]. The properties of the linear regression is related to the addition of data points, the

[3] `https://weka.sourceforge.io/doc.dev/weka/classifiers/bayes/NaiveBayes.html`
[4] `https://javadoc.io/doc/org.deeplearning4j/deeplearning4j-nn/latest/index.html`

rescaling of inputs, the shifting of variables, the reordering of data, and the rotation of independent variables. These MRs were grouped into 6 categories and derived from the properties of the targeted algorithm. For testing a convolutional neural network, we used 11 MRs developed by Ding et al. [24]. The MRs were developed on three levels: system level, data set level, and data item level. The MRs in the data set level is created based on the relation of the classification accuracy of reorganized training data sets. The MRs in the data item level is created based on the relation of the classification accuracy of reproduced individual images.

5.3.5 Source and Follow-up Test Cases

As we described before, MT involves the process of generating source and follow-up test cases based on the MRs used for testing.

IBk uses a training data set to train the k-nearest neighbor classifier, and a test data set is used to evaluate the performance of the trained classifier. After executing the source test case, the output will be the class labels predicted for the test data set, which is a value from the set {0,1,2,3,4}. The attribute values are randomly selected within the range [0, 100]. The values of the class label are randomly selected from the set {0,1,2,3,4,5}. The size of the training testing data sets ranges within [0, 500]. To generate the follow-up test cases using these source test cases, we applied the input transformations described in the MRs.

Linear Regression uses a training data set to train the model, and a test data set is used to evaluate the performance of the trained model. We obtained the training data from the machine learning repository [5] as the source data set. The training data contains 515345 instances and 90 attributes. To generate the follow-up data set using these source data set, we applied the input transformations described in the MRs.

[5]https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD

Similarly, for Naive Bayes, we obtained the training data from the machine learning repository [6] as source data set. The training data contains 48842 instances and 14 attributes. To generate the follow-up data set using these source data set, we applied the input transformations described in the MRs. For testing the convolutional neural network, we obtained the training data [7] as the source data set. The training dataset contains 60000 handwritten digits and 10000 images for testing. To generate the follow-up test cases using these source test cases, we applied the input transformations described in the MRs.

5.3.6 Mutant Generation

For each subject program, we aimed at developing generated mutant set to be used as F_v. For this, we used three automated mutation tools: μJava[8], PIT[9] and Major[10]. For generating mutants using μJava, we used all the method level mutation operators [52]. With PIT mutation tool [23] and the Major mutation tool [41] we used all the available mutation operators provided by the tools. All the mutants that were generated using these tools were *first order mutants*, where a mutation operator is used to make a single modification to the source code to generate the mutant. Table 5.1 shows the number of mutants used for evaluation.

.

[6] https://archive.ics.uci.edu/ml/datasets/Adult
[7] http://yann.lecun.com/exdb/mnist/
[8] https://github.com/jeffoffutt/muJava
[9] http://pitest.org/
[10] http://mutation-testing.org/

Table 5.1: Mutants generated for version v_k of the SUTs

Version	**Subject**	**#Major**	**#PIT**	**# μJava**
3.8.6	IBk	1021	N/A	621
3.8.6	Linear Regression	377	73	N/A
3.8.6	Naive Bayes	452	452	N/A
0.9.1	CNN	92	N/A	N/A

Table 5.2: Validation Setup

Subject	$\mathbf{T_{sp}}$	$\mathbf{F_v}$
IBk	Test1	Major
Linear Regression	YearPrediction	Major
Naive Bayes	Adult	PIT
CNN	MNIST	PIT

5.4 Results and Discussion

In this Section, we discuss our experimental results and provide answers to the two research questions that we listed in Section 5.3. For each subject program, we carried out the validation procedure described in Section 5.3.1 using the setup described in Table 5.2. In this setup, we used the generated mutant set for evaluating the prioritized MR ordering as F_v, and the source test cases as T_{sp}. For example, in Table 5.2 for IBk, Test1 refers to the randomly generated data set that we described in Section 5.3.5. Similarly, for CNN, MNIST refers to the 60000 images used for training and testing. Adult and YearPrediction refer to the training data set used for Naive Bayes and Linear Regression programs respectively. Figure 5.2 shows the

average fault detection effectiveness for evaluation runs described above vs. the MR set size used for testing, for each subject program. We also plot the percentage of faults detected by the random baseline, statement, and branch coverage based MR ordering for comparison. Results for each research question are provided below.

5.4.1 RQ1: Comparison of the MR Prioritization approaches against the Random Baseline

5.4.1.1 Fault detection effectiveness

We formulated the following statistical hypothesis to answer RQ1 in the context of fault detection effectiveness:

H_1: For a given MR set of size m, the fault detection effectiveness of the MR set produced by the Data Diversity-based approach is higher than that of the random baseline.

The null hypothesis H_{0x} for each of the above-defined hypotheses H_X is that the Data diversity-based approaches perform equal to or worse than the random baseline.

In Table 5.3 to 5.6, we list the relative improvement in average fault detection effectiveness for the Data Diversity-based method over the currently used random approach of prioritizing MRs for Linear Regression, Naive Bayes, IBk, and CNN respectively. To evaluate the above hypotheses, we used the paired permutation test since it does not make any assumptions about the underlying distribution of the data [31]. We apply the paired permutation test to each MR set size for each of the subject programs with α= 0.05, and the relative improvements that are significant are marked with a *.

As shown in Table 5.3, for IBk, Rule-based metric shows improvements in average fault detection effectiveness over the random approach for all the MR set sizes except for the last MR set size. Particularly, among the different MR set sizes, the increase in fault detection percentage varies from 1.23% to 29.5%. Therefore, we

reject the null hypotheses H_{01} for IBk for Rule-based, Anomaly, Clustering-based, and Data Distribution-based metrics. Similarly for Linear Regression, Table 5.4 indicates that the Anomaly-based, Clustering-based and Data Distribution-based metrics show consistent significant improvements in fault detection over random. Therefore we can reject H_{01} in general for Regression. The fault detection varies from 2.5% to 30% for the three metrics. Table 5.5 shows the relative improvement in the average fault detection percentage for Naive Bayes. Our proposed metrics shows improvements for MR set sizes $m = 1$ to $m = 5$. Also overall, the relative improvement across MR prioritization methods varies between 0% - 40%. Therefore we reject H_{01} in general for Naive Bayes. Table 5.6 shows the relative improvement in the average fault detection percentage for CNN. Clustering-based metric shows improvements for all MR set sizes except MR set size $m = 5$. Also overall, the relative improvement across MR prioritization methods varies between 0% -66%. Therefore we can reject H_{01} in general for CNN.

Therefore our Data Diversity-based approach outperforms the random approach for all our subject programs.

5.4.1.2 Effective number of MRs used for Testing In Figure 5.2, we show the effective number of MRs using the fault detection thresholds for the subject programs. The pink vertical lines represent effective MR set size when using the Rule-based, green vertical line represents Data Distribution, the brown vertical line represents the Clustering-based and the orange vertical lines represent the same for the random baseline. The respective thresholds are shown near each vertical line.

Similarly for CNN, the Clustering-based approach provides an effective MR set the size of 2 for the 2.5% threshold and the random approach provides an effective MR set size of 4 for the 2.5% threshold. In Naive Bayes, the Anomaly-based approach

Table 5.3: Relative improvement in fault detection effectiveness of the Data Diversity-based approach compared to random baseline for IBk

MR Set-Size	**Rule-based**	**Anomaly-based**	**Clustering-based**	**Data Distribution**
1	29.54%*	29.54%*	25%*	20.45%*
2	17.54%*	8.77%*	5.26%*	19.29%*
3	11.11%*	3.17%*	9.52%*	9.52%*
4	5.88%*	1.47%*	4.41%*	2.94%*
5	5.63%*	4.22%*	7.04%*	2.81%*
6	5.40%*	1.35%*	4.05%*	0%
7	6.57%*	0%	1.31%*	0%
8	5.12%*	0%	0%	0%
9	5%*	1.25%*	-2.5%	3.75%*
10	2.43%*	1.21*	-1.21%	2.43%*
11	0%	0%	0%	0%

provides an effective MR set size of 1 for the 2.5% threshold and the random approach provides an effective MR set size of 3 for the 2.5% threshold. In Linear Regression, the Data Diversity-based approach provides an effective MR set size of 1 for the 2.5% threshold and the random approach provides an effective MR set size of 7 for the 2.5% threshold. This practically leads to 85% reduction in MR set size when compared to the random approach. Therefore for all the subject programs, using our proposed methods resulted in a reduction in the effective MR set size compared to using a random ordering of MRs.

Table 5.4: Relative improvement in fault detection effectiveness of the Data Diversity based approaches compared to random baseline for Linear Regression

MR Set-Size	**Anomaly-based**	**Clustering-based**	**Data Distribution**
1	30.43%*	30.43%*	30.43%*
2	3.38%*	6.77%*	8.47%*
3	-6.15%	1.53%*	9.23%*
4	5.79%*	1.44%*	5.79%*
5	8.33%*	1.38%*	13.88%*
6	6.75%*	1.35%*	10.81%*
7	7.89%*	0%*	7.89%*
8	6.49%*	2.59%*	6.49%*
9	5.12%*	5.12%*	5.12%*
10	2.5%*	2.5%*	2.5%*
11	0%	0%	0%

5.4.1.3 The average time taken to detect a fault Table 5.7 shows the average time taken to detect a fault in Linear Regression and IBk respectively. As shown in these results, using Data Distribution resulted in significant reductions in the average time taken to detect a fault by 26% when compared to a random baseline. Table 5.8 shows the time taken to generate the metrics for the SUT. We observed that the time taken to generate the metrics for the SUT was less than a minute. Therefore, our proposed approach is time and cost-effective when prioritizing MRs.

5.4.1.4 Average percentage of fault detected Table 5.9 shows the APFD for the SUT. We can observe that for all our subject programs, our proposed metrics provided higher APFD values between 0.72-0.92 when compared to random-based

Table 5.5: Relative improvement in fault detection effectiveness of the Data Diversity-based approach compared to random baseline for Naive Bayes

MR Set-Size	**Rule-based**	**Anomaly-based**	**Clustering-based**	**Data Distribution**
1	38.02%*	40.84%*	40.84%*	40.84*
2	8.69%*	8.69%*	8.69%*	8.69*
3	2.04%*	2.04%*	2.04%*	2.04%*
4	1.01%*	1.01%*	1.01%*	1.01*
5	1.01%*	1.01%*	1.01%*	1.01*
6	0%	0%	0%	0%
7	0%	0%	0%	0%
8	0%	0%	0%	0%
9	0%	0%	0%	0%
10	0%	0%	0%	0%

prioritization. As a result, our metrics provide a higher rate of fault detection when compared to the random approach.

5.4.2 RQ2: Comparison of proposed MR Prioritization approach against Code Coverage-based approach

In this research question, we evaluate whether the Data Diversity-based approach outperforms the Code Coverage-based approach. To answer the research question, we formulate the following hypotheses:

H_2: For a given MR set of size m, the fault detection effectiveness of the MR set produced using a Data Diversity-based approach is higher than the fault

Table 5.6: Relative improvement in fault detection effectiveness of the Data Diversity-based approach compared to random baseline for Convolutional Neural Network

MR Set-Size	**Anomaly-based**	**Clustering-based**
1	2.38%*	66.66%*
2	1.81%*	27.27%*
3	1.63%*	14.75%*
4	1.51%*	6.06%*
5	1.44%*	1.44%*
6	1.38%*	-2.77%*
7	1.35%*	8.18%*
8	1.31%*	5.26%*
9	1.29%*	3.89%*
10	1.28%*	2.56%*
11	0%	0%

detection effectiveness of the MR set produced by the Code Coverage-based prioritization.

The null hypothesis H_{0x} for each of the above-defined Hypotheses H_x is that the MR sets generated by the proposed method perform equal to or worse than the MR sets generated by the Code Coverage-based prioritization in terms of fault detection effectiveness.

Table 5.10 and Table 5.11 show the relative improvement in fault detection percentage between the MR sets generated by the Data Diversity-based approach over statement and branch coverage-based approach for IBk. We observed that the Rule-based, anomaly, and Clustering-based metric provided improvement in fault

Table 5.7: Comparison of average time taken to detect a fault using Data Diversity method and Random baseline for Linear Regression and IBk

Subject	**Tsp**	**Data Distribution**	**Rule Based**	**Random Baseline**	**% Time Reduction**
IBk	Test1	-	8910.80s	12562.49s	29.06%
Linear Regression	Prediction	17057.43s	-	21609.766s	26.06%

detection over the statement coverage-based approach for all the MR set sizes except $m = 11$. Therefore we can reject the null hypotheses H_{02}.

Table 5.12 shows the relative improvement in fault detection percentage between the MR sets generated by the Data Diversity-based approach and Code Coverage-based approach for Naive Bayes. We observed that all the metrics provided improvement in fault detection for MR set size $m = 1$ to $m = 4$. Therefore we cannot reject the null hypotheses H_{02} in general for Naive Bayes. Table 5.13 shows the relative improvement in fault detection percentage between the MR sets generated by the Data Diversity-based approach over statement and branch coverage-based approach for Linear Regression. We observed that Anomaly-based and Data Distribution metrics provided improvement in fault detection only for all MR set sizes except $m = 11$. Therefore we reject the null hypotheses H_{02} in general for Linear Regression. Table 5.14 shows the relative improvement in fault detection percentage between the MR sets generated by the Data Diversity-based approach over statement and branch coverage-based approach for CNN. We observed that the Clustering-based approach showed improvement in fault detection for all the MR set sizes and the Anomaly-based approach showed improvement in fault detection only for MR set

Table 5.8: Time taken to generate the metrics for SUT

SUT	**Rule-based**	**Anomaly-based**	**Clustering-based**	**Data Distribution**
IBk	9 sec	7 sec	17 sec	6 sec
Linear Regression	-	4 sec	20 sec	3 sec
Naive Bayes	8 sec	6 sec	18 sec	5 sec
CNN	N/A	200 sec	300 sec	N/A

Table 5.9: Average percentage of fault detected for subject programs

Subject	**Data Distribution**	**Rule Based**	**Clustering Based**	**Anomaly Based**	**Random Based**
IBk	0.85	-	0.91	0.72	0.72
Linear Regression	0.89	0.92	0.97	0.95	0.78
CNN	-	-	0.95	0.95	0.85
Naive Bayes	0.90	0.93	0.95	0.92	0.67

sizes $m = 1$ to $m = 5$. Therefore, we reject the null hypothesis H_{02} in general for Clustering-based metrics. Overall, our proposed metrics outperform the statement and branch coverage-based approach for the subject programs.

Table 5.10: Average relative improvement in fault detection effectiveness of Data Diversity-based approach over statement coverage based approach for IBk

MR Set-Size	**Rule-based**	**Anomaly-based**	**Clustering-based**	**Data Distribution**
1	39.02%*	3.20%*	34.14%*	14.63*
2	36.73%*	26.53%*	22.44%*	6.12%*
3	9.37%*	1.56%*	7.81%*	10.93%*
4	7.46%*	2.98%*	5.97%*	1.49%*
5	8.69%*	7.24%*	10.14%*	0%
6	8.33%*	4.16%*	6.94%*	2.77%*
7	9.45%*	2.70%*	4.05%*	2.70%*
8	7.89%*	2.63%*	2.63%*	2.63%*
9	5.0%*	1.25%*	-2.50%*	3.75%*
10	2.43%*	1.21%	-1.21%	2.43%*
11	0%	0%	0%	0%

5.4.3 RQ3: Comparison of Data Diversity-based MR prioritization approach against the Fault-based prioritization approach

In this research question, we evaluate whether the Data Diversity-based ranking approach outperforms Fault-based approaches. To answer the research question, we formulate the following hypotheses:

H_3: For a given MR set of size m, the fault detection effectiveness of the MR set produced using Data Diversity-based is higher than the fault detection effectiveness of the MR set produced by the Fault-based prioritization.

The null hypothesis H_{0x} for each of the above-defined Hypotheses H_x is that

Table 5.11: Average relative improvement in fault detection effectiveness of Data Diversity-based approach over branch coverage based approach for IBk

MR Set-Size	**Rule-based**	**Anomaly-based**	**Clustering-based**	**Data Distribution**
1	26.66%*	26.66%*	22.22*	22.22%*
2	36.73%*	26.53%*	22.44%*	6.12%*
3	11.11%*	3.17%*	9.52%*	9.52%*
4	5.88%*	1.47%*	4.41%*	2.49%*
5	5.63%*	4.22*	7.04%*	2.81%*
6	6.84%*	2.73%*	5.47%*	1.36%*
7	8%*	1.33%*	2.66%*	1.33%
8	6.49%*	1.29%*	1.29%*	1.29%
9	6.32%*	2.53%*	-1.26%	5.06%*
10	2.43%*	1.21%*	-1.21%*	2.43%*
11%	0%	0%	0%	0%

the MR sets generated by the Data Diversity-based method perform equal to or worse than the MR sets generated by the Fault-based prioritization in terms of fault detection effectiveness.

Table 5.15 shows the relative improvement in fault detection percentage between the MR sets generated by the Data Diversity-based approach and Fault-based approach for IBk. We observed that the Rule-based metric provided improvement in fault detection over Fault-based approach for all the MR set sizes except $m = 3$, $m = 4$, $m = 10$ and $m = 11$. Therefore, we can reject the null hypothesis H_3 for Rule-based metric except for MR set sizes $m = 3$, $m = 4$, $m = 10$ and $m = 11$. Similarly, the Anomaly-based method did not show improvement over the Fault-based except

Table 5.12: Average relative improvement in fault detection effectiveness of Data Diversity-based approach over statement and branch coverage based approach for Naive Bayes

MR Set-Size	**Rule-based**	**Anomaly-based**	**Clustering-based**	**Data Distribution**
1	40%*	42.85%*	42.85*	42.85%*
2	9.80%*	9.89%*	9.89%*	9.898%*
3	2.04%*	2.04%*	2.04%*	2.04%*
4	1.01%	1.01%	1.01%	1.01%
5	0%	0%	0%	0%
6	0%	0%	0%	0%
7	0%	0%	0%	0%
8	0%	0%	0%	0%
9	0%	0%	0%	0%
10	0%	0%	0%	0%

for MR set size $m = 1$. Therefore, we can reject the null hypothesis H_3 for Anomaly-based metrics. Clustering-based metric showed improvement over Fault-based for all the MR set sizes except $m = 9$, $m = 10$ and $m = 11$. Therefore, we can reject the null hypothesis H_3 for Clustering-based metric. Also Data Distribution metric showed improvement over Fault-based for MR set sizes $m = 6$ to $m = 11$. Therefore we cannot we can reject the null hypothesis H_3 for MR set sizes $m = 6$ to $m = 11$. Therefore, Clustering-based metric outperforms Fault-based approach. Rule-based and Data Distribution metric performs closer to Fault-based approach.

Table 5.16 shows the relative improvement in fault detection percentage between the MR sets generated by the Data Diversity-based approach and Fault-based ap-

Table 5.13: Average relative improvement in fault detection effectiveness of Data Diversity-based approach over statement and branch coverage based approach for Linear Regression

MR Set-Size	**Anomaly-based**	**Clustering-based**	**Data Distribution**
1	33.33%*	30.43%*	33.33%*
2	6.71%*	6.77%*	8.47%*
3	1.53%*	1.53%*	9.23%*
4	1.44%*	1.44%*	5.79%*
5	1.38%*	1.38%*	13.88%*
6	1.35%*	1.35%*	10.81%*
7	0%	0%	7.89%*
8	2.59%*	2.59%*	6.49%*
9	3.79%*	5.12%*	3.79%*
10	2.5%*	2.5%*	2.58%*
11	0%	0%	0%

proach for Linear Regression. We observed that Anomaly-based showed improvement over linear regression for all MR set sizes. Therefore, we can reject the null hypothesis H_3 for Anomaly-based. Similarly clustering and Data Distribution-based metric did not show improvement over Fault-based for all MR set sizes. Therefore, we cannot reject the null hypothesis H_3. Results indicate that the Anomaly-based metric outperforms the Fault-based approach and the data-distribution metric performs equally with the Fault-based approach.

Table 5.17 shows the relative improvement in fault detection percentage between the MR sets generated by the Data Diversity-based approach and Fault-based approach for Naive Bayes. We observed that all the metrics provided no improvement

Table 5.14: Average relative improvement in fault detection effectiveness of Data Diversity-based approach over statement and branch coverage based approach for Convolutional Neural Network

MR Set-Size	**Anomaly-based**	**Clustering-based**
1	86.95%*	204.38%*
2	1.81%*	27.27%*
3	0%	12.90%*
4	1.51%*	6.06%*
5	0%	0%
6	1.38%*	-2.77%
7	1.35%*	8.10%*
8	1.31%*	5.26%*
9	0%	2.56%*
10	0%	1.26%*
11	0%	0%

in fault detection for all the MR set sizes. Therefore we cannot reject the null hypotheses H_{03} in general for Naive Bayes. Also, results indicate that the Data Diversity approach performs equal to the Fault-based approach.

Table 5.18 shows the relative improvement in fault detection percentage between the MR sets generated by the Data Diversity-based approach and Fault-based approach for CNN. We observed that the metrics did not provide improvement in fault detection for the MR set sizes except for MR set sizes $m = 1$. Therefore we cannot reject the null hypotheses H_{03} in general for CNN.

Table 5.15: Average relative improvement in fault detection effectiveness of Data Diversity-based approach over Fault-based approach for IBk

MR Set-Size	**Rule-based**	**Anomaly-based**	**Clustering-based**	**Data Distribution**
1	3.63%*	3.61%*	34.14%*	-22.22%*
2	3.07%*	-4.51%*	22.44%*	-6.12%*
3	-2.77%*	-9.72%*	7.81%*	-9.52%*
4	-2.70%*	-6.75%*	5.97%*	-2.94%*
5	1.35%*	0%	10.14%*	-2.81%*
6	4%*	0%	6.94%*	1.36%*
7	5.19%*	-1.29%*	4.05%*	1.33%*
8	1.23%*	-3.70%*	2.63%*	1.29%*
9	0%	-3.57%*	-2.5%	5.06%*
10	0%	-1.29%*	-1.21%	2.43%*
11	0%	0%	0%	0%

5.4.4 RQ4: Comparison of Data Diversity-based MR prioritization approach against the Optimal ordering

In this research question, we evaluate whether the Optimal ordering outperforms the Data Diversity-based ranking. To answer the research question, we formulate the following hypotheses:

H_4: For a given MR set of size m, the fault detection effectiveness of the MR set produced using Optimal ordering is higher than the fault detection effectiveness of Data Diversity-based prioritization.

The null hypothesis H_{0x} for each of the above-defined hypotheses H_x is that

Table 5.16: Average relative improvement in fault detection effectiveness of Data Diversity-based approach over Fault-based approach for Linear Regression

MR Set-Size	**Anomaly-based**	**Clustering-based**	**Data Distribution**
1	30.43%*	0%	0%
2	3.38%*	-1.56%*	-11.11%*
3	-6.15%	-7.04%*	-7.79%*
4	5.79%*	-4.10%*	-8.75%*
5	8.33%*	-10.97%*	0%
6	6.75%*	-8.53%*	0%
7	7.89%*	-7.31%*	0%
8	6.49%*	-3.65%	0%
9	5.12%*	0%	0%
10	2.5%*	0%	0%
11	0%	0%	0%

the MR sets generated by the Optimal ordering perform to or worse than the MR sets generated by the Data Diversity-based prioritization in terms of fault detection effectiveness.

Table 5.19 shows the relative improvement in fault detection percentage between the MR sets generated by the Data Diversity-based approach and Optimal MR ordering for IBk. Optimal ordering shows improvement for all the MR set sizes except $m = 9$, $m = 10$ and $m = 11$. Therefore, we can reject the null hypothesis H_4. Similarly, Anomaly-based metric shows improvement over Optimal ordering. Therefore, we cannot the reject null hypothesis H_4 for Anomaly-based metrics. The results also indicate that Rule-based, Clustering, and Data Distribution metric provided relative improvement in fault detection effectiveness of less than 12% for

Table 5.17: Average relative improvement in fault detection effectiveness of Data Diversity-based approach over Fault-based approach for Naive Bayes

MR Set-Size	**Rule-based**	**Anomaly-based**	**Clustering-based**	**Data Distribution**
1	-2%	0%	0%	0
2	0%	0%	0%	0
3	0%	0%	0%	0
4	0%	0%	0%	0
5	0%	0%	0%	0
6	0%	0%	0%	0
7	0%	0%	0%	0
8	0%	0%	0%	0
9	0%	0%	0%	0
10	0%	0%	0%	0

MR set sizes $m = 2$ to $m = 11$ and performs closer to the Variable-based approach.

Table 5.20 shows the relative improvement in fault detection percentage between the MR sets generated by the Data Diversity-based approach and Optimal MR ordering for Linear Regression. For Anomaly-based, Optimal ordering shows improvement only for MR set sizes $m = 2$ to $m = 6$ and we can reject the null hypothesis H_4 only for MR set sizes $m = 2$ to $m = 6$. For Clustering-based, Optimal ordering shows improvement only for MR set sizes $m = 2$ to $m = 8$ and we can reject the null hypothesis H_4 only for MR set sizes $m = 2$ to $m = 8$. Similarly, for Data Distribution, Optimal ordering shows improvement only for MR set sizes $m = 2$ to $m = 4$. Therefore, our Data Diversity-based approach performs closer to the Optimal approach.

Table 5.18: Average relative improvement in fault detection effectiveness of Fault-based approach over Data Diversity-based approach for Convolutional Neural Network

MR Set-Size	**Anomaly-based**	**Clustering-based**
1	62.79%*	0%
2	25%*	14.28%*
3	29.03%*	14.28%*
4	19.40%*	14.28%*
5	14.28%*	14.28%*
6	9.58%*	14.28%*
7	6.66%*	0%
8	3.89%*	0%
9	2.56%*	0%
10	1.26%	0%
11	0%	0%

Table 5.21 shows the relative improvement in fault detection percentage between the MR sets generated by the Data Diversity-based approach and Optimal MR ordering for Naive Bayes. Our results indicate that the Data Diversity-based approach performs equal to the Optimal ordering for all MR set sizes. Therefore, we cannot reject the null hypothesis H_4 for Naive Bayes.

Table 5.22 shows the relative improvement in fault detection percentage between the MR sets generated by the Data Diversity-based approach and Optimal MR ordering for Convolutional Neural Network. Optimal ordering shows improvement over Anomaly-based for all MR set sizes and null hypothesis H_4 cannot be rejected. In comparison, Optimal ordering shows improvement over Clustering-based only on

Table 5.19: Average relative improvement in fault detection effectiveness of Optimal ordering over Data diversity-based approach for IBk

MR Set-Size	**Rule-based**	**Anomaly-based**	**Clustering-based**	**Data Distribution**
1	3.50%*	-28.07%*	34.14%*	57.14*
2	5.97%*	-20.96%*	22.44%*	41.30%*
3	10%*	-1.53%*	7.81%*	26.31%*
4	12.5%*	-2.98%*	5.97%*	12.12%*
5	9.33%*	-6.75%*	10.14%*	7.24%
6	6.41%*	4%*	6.94%*	1.35%*
7	3.70%*	-2.63%*	4.05%*	1.31%*
8	2.43%*	-2.56%*	2.63%*	3.84%*
9	0%*	-1.23%*	-2.50%*	1.20%*
10	0%*	-1.20%	-1.21%	0%*
11	0%	0%	0%	0%

MR set sizes $m = 1$ to $m = 4$. Therefore, the Clustering-based metric performs closer to the Optimal ordering for CNN.

5.5 Threat to Validity

External Validity: Our proposed metric is applied to supervised ML classifiers. Metrics such as Rule-based is not suitable for unsupervised classifiers and data-distribution metric can be applied only to data set with numerical values. We plan to design more metrics to cover unsupervised classifiers and non-numerical data. In this work, all the experiment subjects are implementation of machine learning

Table 5.20: Average relative improvement in fault detection effectiveness of Optimal ordering over Data Diversity-based approach for Linear Regression

MR Set-Size	**Anomaly-based**	**Clustering-based**	**Data Distribution**
1	0%	0%	0%
2	18.03%*	14.28%*	12.5%*
3	26.22%*	16.66%*	8.45%*
4	9.58%*	14.28%*	9.58%*
5	5.12%*	12.32%*	0%
6	3.79%*	9.33%*	0%
7	0%	7.89%*	0%
8	0%	3.79%*	0%
9	0%	0%	0%
10	0%	0%	0%
11	0%	0%	0%

algorithm used in the Weka ML library. Although they are popular algorithms and widely used, our findings may not be generalizable to commercial machine learning projects such as autonomous vehicles. To mitigate this threat, we plan to explore the effectiveness of our proposed approach on AI systems in the health care domain and Apollo autonomous driving platform projects in the future.

Internal Validity: We used a dataset having an average of 500000 instances and 90 attributes on our subject programs. However, the dataset size could be small for generating the proposed metrics and prioritizing the MRs. We used less than 100 mutants for validating our prioritized MR ordering for CNN program. However, the number of mutants used may be low.

Table 5.21: Average relative improvement in fault detection effectiveness of Optimal approach over Data Diversity based approach for Naive Bayes

MR Set-Size	**Rule-based**	**Anomaly-based**	**Clustering-based**	**Data Distribution**
1	2.04%	0%	0%	0%
2	0%	0%	0%	0%
3	0%	0%	0%	0%
4	0%	0%	0%	0%
5	0%	0%	0%	0%
6	0%	0%	0%	0%
7	0%	0%	0%	0%
8	0%	0%	0%	0%
9	0%	0%	0%	0%
10	0%	0%	0%	0%

5.6 Summary

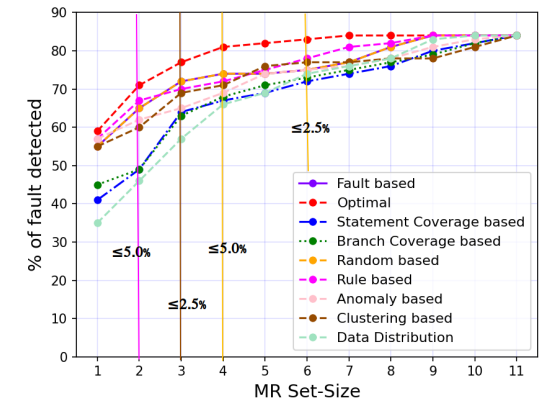
Metamorphic relations in metamorphic testing have multiple source and follow-up test cases. As a result, the execution and training of a machine learning model with a large source and follow-up data set can exponentially increase the execution time and cost of metamorphic testing for ML applications. To overcome this problem, we developed four metrics based on diversity between the source and follow-up data set for the prioritization of MRs.

We evaluated our proposed metrics on four open-source machine learning programs. The experimental results show that utilizing MR prioritization on ML applications can increase the fault detection effectiveness of a given MR set up to 33%

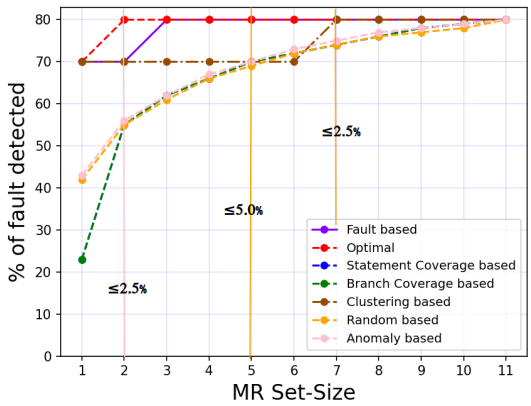
Table 5.22: Average relative improvement in fault detection effectiveness of Optimal approach over Data Diversity-based approach for Convolutional Neural Network

MR Set-Size	**Anomaly-based**	**Clustering-based**
1	2.27%	200%*
2	20.45%*	51.42%*
3	20.45%*	32.5%*
4	20.45%*	1.92%*
5	20.45%*	0%
6	20.45%*	0%
7	20.45%*	0%
8	12.76%*	0%
9	12.76%*	0%
10	0%	0%
11	0%	0%

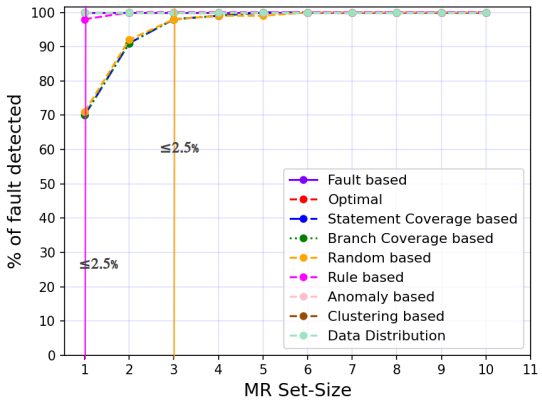
for linear regression, 86% for CNN, and up to 40% for Naive Bayes. In particular, using our proposed MR prioritization approaches reduced the average time taken to detect a fault by 30% for KNN classifier implementation when compared to the most common approach of randomly executing the MRs by 30%. Further, our results show that Data Diversity-based metrics outperform the Code Coverage-based approach for prioritizing MRs by 40%. Our proposed metrics also provided an average 26% higher rate of fault detection when compared to random-based prioritization. Our finding also indicates that the number of MRs needed to execute and test ML programs was reduced by 66% which translates to saving cost and time for software testing practitioners.



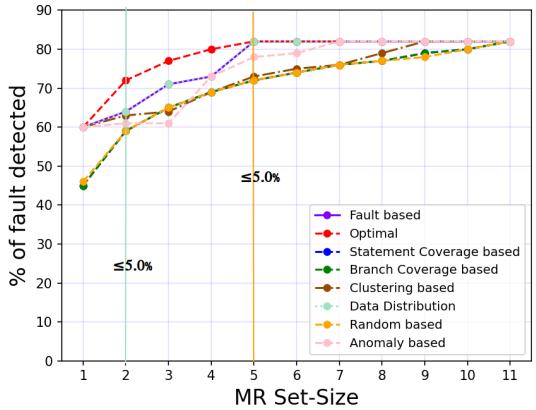
(a) IBk



(b) CNN



(c) Naive Bayes



(d) Linear Regression

Figure 5.2: Fault detection effectiveness for IBk, Linear Regression, Naive Bayes and Convolutional Neural Network

CHAPTER SIX

CONCLUSION

This Chapter first highlights the key takeaway from this dissertation and then presents opportunities for future work.

6.1 Key Takeaway

In Chapter 3, we described an approach to prioritize MRs in the context of regression testing. The prioritization of MRs was based on statement and branch coverage and mutant kill/alive information of the MRs obtained from the previous version of the program. The prioritized MR ordering is used for testing the next version of the program. Our approach reduced the cost of executing MRs and improved fault detection effectiveness while conducting regression testing.

In Chapter 4, we found that individual statements in the program show different levels of importance and centrality to the program output. So, we designed Statement Centrality-based metrics to qualitatively measure the dissimilarity in the execution of source and follow-up test cases of MR to the centrality of the statements in the program. The metrics such as statement affected, statement impacted and fault propagation was used to prioritize the MRs. The metrics provide an improvement over the Code Coverage-based MR prioritization and are performed closer to the Fault-based approach.

Generally, program output is used to check whether the program is behaving as expected. In this work, we proposed a Variable-based approach that uses internal variable values in the program to identify diversity in the execution of the source and follow-up test cases of the MR and prioritize the MRs. This work provides

improvement over the Statement Centrality-based approach and removes the need for code coverage information to prioritize MRs.

In Chapter 5, we proposed Data Diversity-based metrics to prioritize MRs for machine learning programs. The approach identifies the diversity between the source and follow-up training data set in a MR. The result indicated that the greater the diversity between the source and follow-up data set, the greater the fault detection ability of the MR. The approach reduced the number of MRs needed for execution and provided an increase in fault detection effectiveness when compared to Code Coverage-based and random-based execution of MRs.

The key takeaway from this dissertation can be summarized as follows: first, Fault-based and Code Coverage-based approaches provided improved fault detection effectiveness by up to 266% and the time taken to detect a fault is reduced by 40% when compared to random-based prioritization of MRs. Second, the Statement Centrality-based prioritization approach showed that qualitative analysis based on the centrality of statements in the program lead to an increase in fault detection up to 215% over random prioritization and up to 30% over Code Coverage-based prioritization. Also, the average time taken to detect a fault was reduced by 76%, and approaches provided an average 19% higher rate of fault detection over random prioritization of MRs. Similarly, the Variable-based approach based on internal variable values showed higher fault detection effectiveness up to 31% over the Code Coverage-based approach and showed an average 21% higher rate of fault detection over random prioritization of MRs. Finally, Data Diversity-based approach provided an increase in fault detection effectiveness of a given MR set up to 86% over the Code Coverage approach for CNN implementation. The number of MRs needed for executing the ML program was reduced by 66% and the time taken to detect a fault was reduced by 29% over the random approach.

REFERENCES CITED

[1] H. Agrawal and J. R. Horgan. Dynamic program slicing. *ACM SIGPlan Notices*, 25(6):246–256, 1990.

[2] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, Jan 1991.

[3] C. ai Sun, B. Liu, A. Fu, Y. Liu, and H. Liu. Path-directed source test case generation and prioritization in metamorphic testing. *Journal of Systems and Software*, 183:111091, 2022.

[4] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.

[5] P. Ammann and J. Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

[6] M. Asrafi, H. Liu, and F.-C. Kuo. On testing effectiveness of metamorphic relations: A case study. In *2011 Fifth International Conference on Secure Software Integration and Reliability Improvement*, pages 147–156. IEEE, 2011.

[7] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004.

[8] V. K. Ayyadevara. Basics of machine learning. In *Pro Machine Learning Algorithms*, pages 1–15. Springer, 2018.

[9] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, 41(5):507–525, 2015.

[10] D. Berrar. Bayes' theorem and naive bayes classifier. *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, 403, 2018.

[11] D. W. Binkley and K. B. Gallagher. Program slicing. *Advances in computers*, 43:1–50, 1996.

[12] G. Bonaccorso. *Machine learning algorithms*. Packt Publishing Ltd, 2017.

[13] H. B. Braiek and F. Khomh. On testing machine learning programs. *Journal of Systems and Software*, 164:110542, 2020.

[14] Y. Cao, Z. Q. Zhou, and T. Y. Chen. On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions. In *2013 13th International Conference on Quality Software*, pages 153–162. IEEE, 2013.

[15] W.-L. Chao. Machine learning tutorial. *Digital Image and Signal Processing*, 2011.

[16] J. Chen, Y. Bai, D. Hao, L. Zhang, L. Zhang, B. Xie, and H. Mei. Supporting oracle construction via static analysis. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 178–189, 2016.

[17] T. Y. Chen, J. Feng, and T. Tse. Metamorphic testing of programs on partial differential equations: a case study. In *Proceedings 26th Annual International Computer Software and Applications*, pages 327–333. IEEE, 2002.

[18] T. Y. Chen, D. Huang, T. Tse, and Z. Q. Zhou. Case studies on the selection of useful relations in metamorphic testing. In *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004)*, pages 569–583. Polytechnic University of Madrid, 2004.

[19] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. Tse, and Z. Q. Zhou. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys (CSUR)*, 51(1):4, 2018.

[20] T. Y. Chen, F.-C. Kuo, W. Ma, W. Susilo, D. Towey, J. Voas, and Z. Q. Zhou. Metamorphic testing for cybersecurity. *Computer*, 49(6):48–55, 2016.

[21] T. Y. Chen, F.-C. Kuo, T. Tse, and Z. Q. Zhou. Metamorphic testing and beyond. In *Eleventh Annual International Workshop on Software Technology and Engineering Practice*, pages 94–100. IEEE, 2003.

[22] H. Cleve and A. Zeller. Locating causes of program failures. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 342–351. IEEE, 2005.

[23] H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque. Pit: a practical mutation testing tool for java. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 449–452. ACM, 2016.

[24] J. Ding, X. Kang, and X.-H. Hu. Validating a deep learning framework by metamorphic testing. In *2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET)*, pages 28–34. IEEE, 2017.

[25] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering*, 28(2):159–182, 2002.

[26] S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky. Selecting a cost-effective test case prioritization technique. *Software Quality Journal*, 12(3):185–210, 2004.

[27] K. Gallagher and D. Binkley. Program slicing. In *2008 Frontiers of Software Maintenance*, pages 58–67. IEEE, 2008.

[28] G. Gay, M. Staats, M. Whalen, and M. P. Heimdahl. Automated oracle data selection support. *IEEE Transactions on Software Engineering*, 41(11):1119–1137, 2015.

[29] E. Giannoulatou, S.-H. Park, D. T. Humphreys, and J. W. Ho. Verification and validation of bioinformatics software without a gold standard: a case study of bwa and bowtie. *BMC bioinformatics*, 15(16):S15, 2014.

[30] J. Glenday. Ai-powered genderify platform shut down after bias-based backlash.

[31] P. Good. *Permutation tests: a practical guide to resampling methods for testing hypotheses*. Springer Science & Business Media, 2013.

[32] F. Harel-Canada, L. Wang, M. A. Gulzar, Q. Gu, and M. Kim. Is neuron coverage a meaningful measure for testing deep neural networks? In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 851–862, 2020.

[33] M. Harman and R. Hierons. An overview of program slicing. *software focus*, 2(3):85–92, 2001.

[34] P. Harrington. *Machine learning in action*. Simon and Schuster, 2012.

[35] Z.-W. Hui, S. Huang, H. Li, J.-H. Liu, and L.-P. Rao. Measurable metrics for qualitative guidelines of metamorphic relation. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 3, pages 417–422. IEEE, 2015.

[36] L. Igual and S. Seguí. Supervised learning. In *Introduction to Data Science*, pages 67–96. Springer, 2017.

[37] G. Jahangirova. Oracle problem in software testing. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 444–447, 2017.

[38] D. C. Jarman, Z. Q. Zhou, and T. Y. Chen. Metamorphic testing for adobe data analytics software. In *2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET)*, pages 21–27. IEEE, 2017.

[39] Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering*, 37(5):649–678, 2010.

[40] L. Jiang, D. Wang, Z. Cai, and X. Yan. Survey of improving naive bayes for classification. In *International conference on advanced data mining and applications*, pages 134–145. Springer, 2007.

[41] R. Just. The major mutation framework: Efficient and scalable mutation analysis for java. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 433–436. ACM, 2014.

[42] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser. Are mutants a valid substitute for real faults in software testing? In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 654–665. ACM, 2014.

[43] U. Kanewala and J. M. Bieman. Testing scientific software: A systematic literature review. *Information and Software Technology*, 56(10):1219 – 1232, 2014.

[44] P. Kim. Convolutional neural network. In *MATLAB deep learning*, pages 121–147. Springer, 2017.

[45] B. Korel and J. Laski. Dynamic program slicing. *Information processing letters*, 29(3):155–163, 1988.

[46] S. Liao. Chinese facial recognition system mistakes a face on a bus for a jaywalker.

[47] C. C. Liem and A. Panichella. Oracle issues in machine learning and where to find them. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pages 483–488, 2020.

[48] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen. How effectively does metamorphic testing alleviate the oracle problem? *IEEE Transactions on Software Engineering*, 40(1):4–22, 2013.

[49] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen. How effectively does metamorphic testing alleviate the oracle problem? *IEEE Transactions on Software Engineering*, 40(1):4–22, 2014.

[50] P. Loyola, M. Staats, I.-Y. Ko, and G. Rothermel. Dodona: automated oracle data set selection. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 193–203, 2014.

[51] Q.-H. Luu, M. F. Lau, S. P. Ng, and T. Y. Chen. Testing multiple linear regression systems with metamorphic testing. *Journal of Systems and Software*, 182:111062, 2021.

[52] Y.-S. Ma, J. Offutt, and Y.-R. Kwon. Mujava: a mutation system for java. In *Proceedings of the 28th international conference on Software engineering*, pages 827–830. ACM, 2006.

[53] L. Madeyski and N. Radyk. Judy–a mutation testing tool for java. *IET software*, 4(1):32–42, 2010.

[54] A. G. Malishevsky, J. R. Ruthruff, G. Rothermel, and S. Elbaum. Cost-cognizant test case prioritization. Technical report, Citeseer, 2006.

[55] S. Mani, A. Sankaran, S. Tamilselvam, and A. Sethi. Coverage testing of deep learning models using dataset characterization. *arXiv preprint arXiv:1911.07309*, 2019.

[56] D. Marijan, A. Gotlieb, and M. K. Ahuja. Challenges of testing machine learning based systems. In *2019 IEEE international conference on artificial intelligence testing (AITest)*, pages 101–102. IEEE, 2019.

[57] J. Mayer and R. Guderlei. An empirical study on the selection of good metamorphic relations. In *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, volume 1, pages 475–484. IEEE, 2006.

[58] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[59] G. J. Myers, C. Sandler, and T. Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[60] S. Nakajima. Generalized oracle for testing machine learning computer programs. In *International Conference on Software Engineering and Formal Methods*, pages 174–179. Springer, 2017.

[61] S. Nakajima. Dataset diversity for metamorphic testing of machine learning software. In *International Workshop on Structured Object-Oriented Formal Language and Method*, pages 21–38. Springer, 2018.

[62] S. Nakajima and H. N. Bui. Dataset coverage for testing machine learning computer programs. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, pages 297–304, 2016.

[63] A. Ohnsman. Lidar maker velodyne 'baffled'by self-driving uber's failure to avoid pedestrian.

[64] K. O'Shea and R. Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[65] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman. Mutation testing advances: an analysis and survey. In *Advances in Computers*, volume 112, pages 275–378. Elsevier, 2019.

[66] D. Parsons, T. Susnjak, and M. Lange. Influences on regression testing strategies in agile software development environments. *Software Quality Journal*, 22(4):717–739, 2014.

[67] T. Qin. Machine learning basics. In *Dual Learning*, pages 11–23. Springer, 2020.

[68] J. C. Reynar and A. Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the fifth conference on Applied natural language processing*, pages 16–19. Association for Computational Linguistics, 1997.

[69] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella. Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering*, 25(6):5193–5254, 2020.

[70] R. H. Rosero, O. S. Gómez, and G. Rodríguez. 15 years of software regression testing techniques—a survey. *International Journal of Software Engineering and Knowledge Engineering*, 26(05):675–689, 2016.

[71] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering*, 27(10):929–948, 2001.

[72] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés. A survey on metamorphic testing. *IEEE Transactions on software engineering*, 42(9):805–824, 2016.

[73] S. Segura and Z. Q. Zhou. Metamorphic testing 20 years later: A hands-on introduction. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pages 538–539. ACM, 2018.

[74] D. Sharma and N. Kumar. A review on machine learning algorithms, tasks and applications. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 6(10):2278–1323, 2017.

[75] H. Spieker and A. Gotlieb. Adaptive metamorphic testing with contextual bandits. *Journal of Systems and Software*, 165:110574, 2020.

[76] M. Srinivasan and U. Kanewala. Metamorphic relation prioritization for effective regression testing. *Softw. Test. Verification Reliab.*, 32(3), 2022.

[77] M. Srinivasan, M. P. Shahri, I. Kahanda, and U. Kanewala. Quality assurance of bioinformatics software: a case study of testing a biomedical text processing tool using metamorphic testing. In *Proceedings of the 3rd International Workshop on Metamorphic Testing*, pages 26–33. ACM, 2018.

[78] C.-a. Sun, H. Dai, H. Liu, and T. Y. Chen. Feedback-directed metamorphic testing. *ACM Transactions on Software Engineering and Methodology*, 2022.

[79] Q. Tao, W. Wu, C. Zhao, and W. Shen. An automatic testing approach for compiler based on metamorphic testing technique. In *2010 Asia Pacific Software Engineering Conference*, pages 270–279. IEEE, 2010.

[80] S. Tolksdorf, D. Lehmann, and M. Pradel. Interactive metamorphic testing of debuggers. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 273–283, 2019.

[81] G. A. Venkatesh. The semantic approach to program slicing. In *Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*, pages 107–119, 1991.

[82] J. M. Voas. Pie: A dynamic failure-based technique. *IEEE Transactions on software Engineering*, 18(8):717, 1992.

[83] H. Wang, Z. Lei, X. Zhang, B. Zhou, and J. Peng. Machine learning basics. *Deep learning*, pages 98–164, 2016.

[84] M. Weiser. Program slicing. *IEEE Transactions on software engineering*, (4):352–357, 1984.

[85] J. A. Whittaker. What is software testing? and why is it so hard? *IEEE software*, 17(1):70–79, 2000.

[86] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *PROCEEDINGS The Eighth International Symposium On Software Reliability Engineering*, pages 264–274. IEEE, 1997.

[87] X. Xie, J. W. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software*, 84(4):544–558, 2011.

[88] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability*, 22(2):67–120, 2012.

[89] T. Yu, X. Qu, M. Acharya, and G. Rothermel. Oracle-based regression test selection. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 292–301. IEEE, 2013.

[90] D. Zhang and J. J. Tsai. *Advances in machine learning applications in software engineering*. Igi Global, 2006.

[91] J. M. Zhang, M. Harman, L. Ma, and Y. Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 2020.

[92] Z. Zhang. Introduction to machine learning: k-nearest neighbors. *Annals of translational medicine*, 4(11), 2016.

[93] Z. Q. Zhou and L. Sun. Metamorphic testing of driverless cars. 2019.

[94] Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey. Metamorphic relations for enhancing system understanding and use. *IEEE Transactions on Software Engineering*, 2018.

[95] Z. Q. Zhou, S. Xiang, and T. Y. Chen. Metamorphic testing for software quality assessment: A study of search engines. *IEEE Transactions on Software Engineering*, 42(3):264–284, 2015.

[96] C. Ziegler. A google self-driving car caused a crash for the first time. 2016.

APPENDIX: METAMORPHIC RELATIONS

A.1 MRs used for testing the subject programs

In this Section, we discuss the MRs used in each of the subject programs.

A.1.1 MRs for BBMap Program

Below is the set of MRs for testing BBMap program, and the MRs were obtained from Giannoulatou et al. [29].

- MR1 (Read Addition): The reads in the initial test are duplicated, making the input for the follow-up test twice as large. We also expect the coverage in the follow-up test to be twice of the original coverage. The MR include one source test case (reads) and one follow-up test case (duplicated reads).
- MR2 (Mapped Reads): After initial mapping, only the mapped reads are selected and remapped against the reference genome. We expect that all of the reads will be mapped. The MR include one source test case(reads) and one follow-up test case (mapped reads).
- MR3 (Correction of errors): After initial mapping, the mapped reads are selected and any mismatch or error is corrected in the reads. We expect the output mapping to remain the same. The MR include one source test case(mapped read) and one follow-up test case(corrected read).
- MR4 (Random permutation of reads): The reads in the FASTQ files are reshuffled. The permutation is the same for Read1 and Read2 reads. We expect the output mapping to be the same as the original output. The MR include one source test case (reads) and one follow-up test case (permuted read).
- MR5 (Quality score increase of reads): After initial mapping, the quality score for all mapped sequences is increased. We expect the output mapping to remain the same. The MR contain one source test case (reads) and one follow-up test case(quality score increased).
- MR6 (Read Removal): Half of the reads from the initial input collection are deleted. We also expect the coverage in the follow-up test to be half of the original coverage. The MR contain one source test case(reads) and one follow-up test case(reads removed).
- MR7 (Reverse Complement of reads): The reads are reverse complemented and the corresponding quality values are reversed. We expect the output mapping to be same as the original output. The MR contains one source test case(Reads) and one follow-up test case(reverse complemented reads).

- MR8 (Unmapped Reads): After initial mapping, only the unmapped reads are selected and remapped against the reference genome. We expect that none of the reads will be mapped. The MR include one source test case(mapped read) and one follow-up test case(unmapped read).

A.1.2 MRs for IBK and Naive Bayes Program

The MRs used for testing IBK and Naive Bayes program is discussed below and the MRs were obtained from Xie et al. [87]

- MR1 (Consistence with affine transformation): The result should be same, if we apply some affine transformation function, f(x) = kx + b,(k$\neq$ 0), to every value x in some subset of features in the training and testing data. The appropriate value of K=3 (i.e.number of nearest neighbours) such that MR becomes a necessary property for IBK. The MR contain one source test case (training and testing set). The MR contains one source and follow-up test case.

- MR2 (Permutation of the attribute): If we permute the m attributes of all the samples and the test data, the result should remain unchanged. The appropriate value of K=3 (i.e.number of nearest neighbours) such that MR becomes a necessary property for IBK. The MR contains one source and follow-up test case.

- MR3 (Addition of uninformative attributes): If we add some new feature that is equally associated with all classes, the predictions of the test data should not be changed. The appropriate value of K=3 (i.e.number of nearest neighbours) such that MR becomes a necessary property for IBK. The MR contains one source and follow-up test case.

- MR4 (Consistence with re-prediction): Suppose we predict some test case t as class l_i . If we append t to our training data and re-create the model, t should still be classified as class l_i. The appropriate value of K=3 (i.e.number of nearest neighbours) such that MR becomes a necessary property for IBK. The MR contains one source and follow-up test case.

- MR5 (Additional training sample): For the source input, suppose we get the result $c_t = l_i$ for the test case t_s. In the follow-up input, we duplicate all samples in S and L which have label l_i. The output of the follow-up test case should still be l_i. The appropriate value of K=3 (i.e.number of nearest neighbours) such that MR becomes a necessary property for IBK. The MR contains one source and follow-up test case.

- MR6 (Addition of classes by re-labeling sample): For some test cases not of class l_i, we switch the class label from x to x . Then every test case predicted as class l_i should still be predicted as class l_i with the re-labeled samples. The appropriate value of K=3 (i.e.number of nearest neighbours) such that MR

becomes a necessary property for kNN. The MR contains one source and follow-up test case.

- MR7 (Permutation of class labels): If we permute the order of the class labels with some random permutation p(l_i) where l_i is a class label, all test cases which were predicted as l_i should now be predicted as p(l_i). The appropriate value of K=1 (i.e.number of nearest neighbours) such that MR becomes a necessary property for IBK. The MR contains one source and follow-up test case.

- MR8 (Addition of informative attribute): If we add some new feature that is strongly associated with one class, l_i, then for every prediction that was class l_i, the prediction with this new attribute should also be class l_i. The appropriate value of K=1 (i.e.number of nearest neighbours) such that MR becomes a necessary property for IBK. The MR contains one source and follow-up test case.

- MR9 (Addition of classes by duplicating samples): Suppose we duplicate every class except for n, and give them all a new class. Then every test case predicted as class l_i should still be predicted as class l_i with the duplicated samples. The appropriate value of K=1 (i.e.number of nearest neighbours) such that MR becomes a necessary property for IBK. The MR contains one source and follow-up test case.

- MR10 (Removal of classes): If we remove some class l_i, the remaining predictions should remain unchanged. The appropriate value of K=1 (i.e.number of nearest neighbours) such that MR becomes a necessary property for IBK. The MR contains one source and follow-up test case.

- MR11 (Removal of samples): If we remove samples that have not been predicted as class l_i, then all cases which were predicted as l_i should remain unchanged. The appropriate value of K=1 (i.e.number of nearest neighbours) such that MR becomes a necessary property for IBK. The MR contains one source and follow-up test case.

A.1.3 MRs for Linear Regression

The MRs used for testing linear regression program is discussed below and the MRs were obtained from Luu et al. [51]

- MR1 (Inserting a predicted point): The regression line will remain the same after being updated by adding a point selected from the line into the original data set. Then we expect follow-up output be same as the source output.

- MR2 (Inserting the Centroid): The linear regression line will remain the same after being updated by adding the centroid into the original data set. Therefore, adding the centroid of data into the source input to form a follow-up input will not change the follow-up estimator for the regression form.

- MR3 (Reflecting the dependent variable): Reflecting the points over a certain x-axis will reflect the regression line over the same axis. The follow-up estimator becomes a reflection of the source estimator when the sign of the dependent variable is reversed in the follow-up input set.

- MR4 (Reflecting an independent variable while keeping the others unchanged): Reflecting the points over the y-axis will reflect the regression line over the same axis. The x-coordinate of an independent variable becomes reflected while the x-coordinate of other independent variables and the y-coordinates remain the same.

- MR5 (Scaling the dependent variable): The source data points are scaled in the y axis by a given factor to generate follow-up data points. Then, we expect the the follow-up output to be scaled proportionally by the same factor.

- MR6 (Scaling an independent variable while keeping the others unchanged): The source data points are scaled in a particular x axis by a given factor to generate follow-up data points. We expect the slope of regression line scaled reciprocally with respect to that axis.

- MR6 (Swapping Samples): Swapping any two data points does not alter the regression hyperplane. Given the source input I_s and source output O_s, suppose that we swap two data points to define the follow-up input I_f and follow-up output O_f. Then, the follow-up output O_f would be the same as the source input O_s.

- MR7 (Swapping two independent variables while keeping the others unchanged): swapping two independent variables points while keeping the others unchanged will only swap the two relevant components of the follow-up estimator.

- MR8 (Rotating two independent variables while keeping the others unchanged): when the points are rotated in the plane perpendicular to the y axis, the regression hyperplane will be rotated. Basically, if we rotate the axes of any two independent variables by an angle, the corresponding components of estimator would be also rotated by the same angle.

- MR9 (Shifting the dependent variable): When the points are shifted by a given distance along the y axis, the regression line will be shifted along this axis by the same distance. Therefore, if a constant is added into the values of dependent variable, the intercept of the new estimator would be increased by the same value.

- MR10 (Shifting an independent variable while keeping the others unchanged): When the points are shifted by a given distance along a certain x axis, the regression line will be shifted in parallel along this axis accordingly. So,

if a constant is added into values of an independent variable, the intercept component of follow-up estimator would be decreased by an amount equal to the product of the constant and the value of the corresponding component of source estimator.

Training Set	Test Set
@attribute pictures numeric	@attribute pictures numeric
@attribute paragraphs numeric	@attribute paragraphs numeric
@attribute files numeric	@attribute files numeric
@attribute files2 numeric	@attribute files2 numeric
@attribute profit {0,2,1,3,4}	@attribute profit {0,2,1,3,4}
@data	@data
45,16,3,38,0	6,42,10,89,0
15,67,89,33,4	5,8,24,67,1
59,87,97,11,0	8,45,56,33,2
86,92,96,15,2	7,89,56,45,0
80,30,96,11,4	78,55,23,35,0
23,14,49,41,1	92,23,45,78,3
94,17,25,15,0	45,6,78,23,1
95,28,99,76,3	55,97,45,23,4
50,92,0,74,2	
33,48,49,95,0	

Figure A.1: Sample Data Set for IBk Program

A.1.4 MRs for LingPipe Program

The MRs used for testing LingPipe program is discussed below and the MRs were modified from Srinivasan et al. [77]. In addition and removal MR for sentence and paragraph, sentence and paragraph boundary is maintained. Sentence boundaries are identified by scanning the text for sequences of characters separated by white space (tokens) containing one of the symbols !, . or ? [68].

- MR1 (Adding a sentence to another sentence): Given two sentences S_1 and S_2, we append S_2 to S_1 to form new sentence S$\prime$ such that the sentence boundary is maintained. We expect the union between the bio-entities of the sentence S_1 and sentence S_2 to be a subset of the bio-entities in the resultant sentence S$\prime$. The MR contains two source test cases (sentence 1 and sentence 2) and one follow-up test case(sentence)

- MR2 (Adding a sentence to a paragraph): Given a sentence S and an index i, we add it to the position i of a paragraph P such it immediately follows sentence boundary. The resultant text of this addition is P$\prime$. We expect the union between the bio-entities of the sentence S and paragraph P to be a subset of the bio-entities in the resultant text P$\prime$. The MR contains two source test cases (sentence and paragraph) and one follow-up test case(paragraph)

- MR3 (Adding a paragraph to an article): Given a paragraph P and an index i, we add this paragraph to the position i of an article A such it immediately

follows paragraph boundary. The resultant text of this addition as A/. We expect the union between the bio-entities of the paragraph P and article A to be a subset of the bio-entities in the resultant text A/. The MR contains two source test cases (Paragraph and article) and one follow-up test case(Article).

- MR4 (Adding a list of random words to another list): Given two list of random words L_1 and L_2, we append L_2 to L_1. The resultant list as L/. We expect the bio-entities of L/ to be the union of bio-entities extracted from L_1 and L_2. The MR contains two source test cases(word list1 and word list2) and follow-up test case (word list3).

- MR5 (Removing a list of random words from a sentence): Given a sentence S, we remove a list of words L from S. The resultant text of this deletion as S/. We expect the bio-entities of S/ to be the subtraction of the bio-entities extracted of L from S. The MR contains two source test case (sentence and word) and one follow-up test case(sentence).

- MR6 (Removing a sentence from a paragraph): Given a paragraph P and an index i, we remove a sentence S, starting from position i, from P such that the sentence boundary is maintained. We expect the difference between the bio-entities of the original paragraph and removed sentence to be a subset of the bio-entities in the resultant paragraph P/. The MR contains two source test case(Paragraph and sentence) and one follow-up test case(Paragraph).

- MR7 (Removing a paragraph from an article): Given an article A, we remove a random paragraph P from A such that the paragraph boundary is maintained. The resultant text of this deletion as A/. We expect the difference between the bio-entities of the article A and removed paragraph to be a subset of the bio-entities in the resultant article A/. The MR contains two source test case(Article and Paragraph) and one follow-up test case(Article).

- MR8 (Removing some words from a list of random words): Given a list of random words L_1, we remove half of these words from the end of file. We refer to the removed part and the remaining part as L_2 and L/. We expect the bio-entities of L/ to be the subtraction of the bio-entities of L_1 and L_2. The MR contains two source test case (list of thousand random words L_1 and list of five hundred words L_2) and one follow-up test case(list of five hundred word L_2).

- MR9 (Shuffling paragraphs of an article): Given an article A, we shuffle all the paragraphs of A. The new resultant text as A/. We expect the bio-entities of the article A to be subset of the resultant text A/, although the position of the bio-entities can vary. The MR contains one source test case (Article) and one follow-up test case (Shuffled Article).

- MR10 (Shuffling a list of random words): Given a list of random words L, we shuffle all words and create a new list of words L. We create a new list of words L′. We expect the bio-entities of L′ to be equal to the bio-entities extracted from L. The MR contains one source test case(random words) and one follow-up test case(shuffled words).