PalArch's Journal of Archaeology of Egypt / Egyptology

IMPROVING MUTANT SELECTION FOR GUI USING SIMILARITY RELATION AND CONDITIONAL ENTROPY

Nurfadhilah Binti Sapingi[1], *Noraini Binti Ibrahim*[2], *Mustafa Mat Deris*[3]

[1,2,3]Faculty of Science and Information Technology, Universiti Tun Hussein Onn Malaysia (UTHM), Parit Raja, Johor, Malaysia

E-mail: [1]fadhilah_sapingi@yahoo.com, [2]noraini@uthm.edu.my, [3]mmustafa@uthm.edu.my

Nurfadhilah Binti Sapingi, Noraini Binti Ibrahim, Mustafa Mat Deris. Improving Mutant Selection For Gui Using Similarity Relation And Conditional Entropy--Palarch's Journal Of Archaeology Of Egypt/Egyptology 17(10), 394-410. ISSN 1567-214x

Keywords: Gui Testing, Software Testing, Mutation Analysis, Mutation Operators, Similarity Relation, Conditional Entropy

ABSTRACT

Mutation technique is known as one of the most powerful techniques to detect fault capabilities. Mutation generates the fault version called mutants. Mutation makes small changes in the code. The output is analysed if it is different from the original program. Then, the mutant can be killed, or otherwise the mutant said as live. This mutation technique will be used on Graphical User Interface (GUI), which plays an important role in testing the interaction between user and software. Using this mutation technique, it may detect more faults on Graphical User Interface (GUI) during testing to produce a good test suite. The problem with the mutation technique is that it is expensive, thus new refinement technique, such as mutant selection, is proposed. The lack of mutant selection decreases the effectiveness and efficiency of testing due to a random selection of a small subset of mutants and reducing some operator mutants from the test set for execution. Regardless of this, reducing the number of mutants for execution is still important. Thus, this paper proposed new refinement techniques for improving mutant selection in terms of effectiveness and efficiency of testing known as Similarity Relation and Conditional Entropy. Similarity Relation classifies the same mutants in the same class to avoid redundancy, while Conditional Entropy selects mutant operator based on the classification of mutants. The results show that Similarity Relation can reduce 85% of mutants, while Conditional Entropy reduces 80% of mutant operator with 100% defect fault capabilities. Similarity Relation and Conditional Entropy can improve mutant selection to select the optimum mutants and mutants' operator for testing without less effectiveness and efficiency.

INTRODUCTION

Graphical User Interface (GUI) is a flexible graphical element for user to interact with software system. Graphical User Interface (GUI) provides user with graphical icons, such as button, edit boxes, and text to show information [1-3]. Graphical User Interface (GUI) is the first expectation from user to measure the quality of system. The disadvantage of the system comes from the user if the Graphical User Interface (GUI) is unable to access and not user-friendly. Regardless of this, the Graphical User Interface also needs a testing to establish connection between software and its end user as GUI is the only way for user to access the functionality of the software and reliable Human Computer Interaction (HCI) process.

Graphical User Interface testing is more complex than other testing methods [3, 4] and to measure accurateness and quality, the Graphical User Interface (GUI) becomes more complex. Applying adequate testing is the only way to help increase the confidence of user on the correctness of any application. There are several testing techniques for testing Graphical User Interface (GUI), but they still lack in detecting fault capabilities [5-7]. To develop a high quality testing, the only approach that can help is a test that detects faults. Concerned about this issue, the mutation testing concept is applied to obtain the quality of Graphical User Interface (GUI).

The mutation concept is a powerful technique for detecting fault in the code for getting the quality of test suite [8-10] [19-20]. Mutation generates fault versions called as mutants on a system being tested. Mutation makes small changes on the source code and analysis of the output. If the output has different results from the original system, then the result can be said as a killed mutant, or otherwise the result can be said as alive mutants.

Mutation has a famous problem, which is being expensive. Mutation testing generates a large number of mutants even for a small program. It becomes costly due to compiling and executing the large number of copies of a program. Concerned about this problem, many researchers introduced the refinement method of mutation to solve this issue. The well-known method of reduction, which inadvertently removes mutants and to be applied in implementation, will select only a subset of mutations, and usually they do not take into account the structure of the system and mutants [11-16]. The lack of percentage decreases the effectiveness of the test. Nevertheless, there is still a need for assessing the reduced number of mutants of these new methods, which are proposed to achieve Similarity Relation and Conditional Entropy.

Similarity Relation and Conditional Entropy are applied in the concept of mutations to decrease the number of mutants by classifying them into same class, while Conditional Entropy selects mutant operators depend on mutant classification. This method will cut down the number of mutants without abbreviating the effectiveness of random mutant removal and avoid the advantage of mutants to be implemented again.

In this context, this research attempts to introduce a new technique using mathematical approach, which classifies the same attributes in one class to avoid redundancy for executing the test set. To identify the same attributes, firstly, the data set is transformed into matrix table and classified using the formula of Similarity Relation. When the mutants are classified in the same class, then the formula of Conditional Entropy is used to select the mutants' operator. The results show that this proposed technique produces a good percentage in the reduction of mutants with 100% defect fault capabilities. Thus, this experiment showed that the proposed technique can improve the mutants' selection.

The main contributions of this paper are:
a. To propose a method to select optimum mutants using similarity Relation.
b. To evaluate and compare proposed approaches with other mutation test approaches (randomly selected) in terms of effectiveness and efficiency.

This remaining part is organised as follows: Part 2 will introduce the material and method. Part 3 discusses regarding results of experiment. Part 4 concludes the aims of the research.

MATERIAL AND METHOD

This part highlights several refinement mutation techniques from previous works, the process of mutation, and describes the construction of Similarity Relation and Conditional Entropy for mutation.

Mutation technique

Mutation defines as an effective method or technique to identify fault and obstacles of identifying the adequate test data [8]. The number to detect potential fault is massive and it is too much to generate all mutants from one complete system.

Traditional Mutation is the famous one previously the refinement mutation was introduced [8]. The traditional mutation technique only point a group of fault, which is next to the precise version based on that program and it is hoped to be enough to reproduce all the faults [8]. From previous works [8], the mutants used in traditional mutation are limited to simple mutants only.

Mutation requires an expensive cost due to the large number to generate mutants and execute against the mutants. Many researchers are concerned about this issue and tried to solve it by introducing the mutant reduction technique. The techniques are [8]: Mutants Sampling, Mutants Clustering, Higher Order Mutation and Selective Mutation.

Mutant sampling

Mutant sampling is a not difficult approach by Acree [8] and Budd [8]. For the first step, all the potential mutants are produced as a traditional mutation. The second step is the mutants being selected randomly and the remaining mutants

are discarded. Based on discussions among researchers [8], this method is agreed to be less effective in terms of mutation score due to the discarded mutants from a full set of mutants.

Mutant clustering

Mutant clustering was recommended by Hussains's [8]. It is done by clustering the mutants using the clustering algorithm. The first step generates all possible mutants. The second step applies the algorithm to group the first order mutants into various clusters based on the killable of the test cases. The similar set in the same cluster will be killed. Only a small subset from the cluster will be used and others are discarded. The empirical study indicates that the clustering mutation is able to select fewer mutants, but at the same time, still maintains the mutation score.

Selective mutation

Selective mutation was proposed by Mrthur [8] and extended by offut [8]. This method diminishes the amount number of mutants based on the operator applied. The basic idea is that it finds a small set of mutation operators that generate the subset of all the possible mutants without losing the effectiveness. This method will impact the redundant data due to the mutation operator that generates various numbers of mutants and some mutation operators are able to generate more mutants as well. Researchers have also discussed on [8] how this method reduces the number of equivalent mutants while maintaining the effectiveness. This approach also achieves the highest rate of reduction method compared to other methods.

Higher order mutation

Higher Order mutation was recommended by Jia and Harman [8], researcher apply the concept for subsuming the HOMs, and this way is prefer to replace the FOMs with the single HOMs to reduce the amount of mutants. This idea come out from traditional mutants technique which is FOMs was created when apply the mutation operator just once time, while the HOMs was created when apply the mutation operator more than once time.

Process of mutation

The first process in Mutation is based on the traditional method. Fig. 1 illustrates the flow of mutation working. From the program, the set of fault program is labelled as '***P***' and called as mutants labelled as '*P*' that was generated from the original program

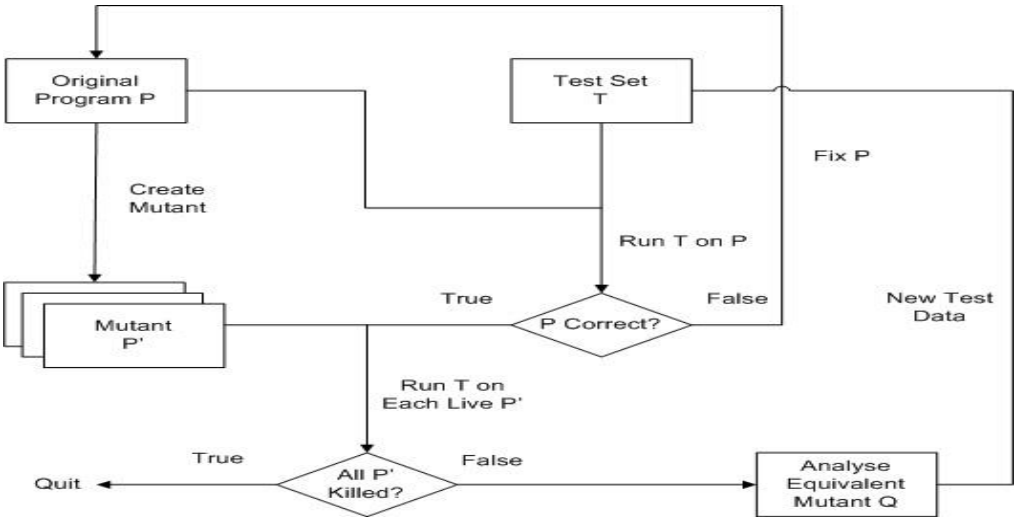

Figure 1: Process of Mutation Analysis [9]

The mutation operators are designed to modify the variable and expression by insertion, replacement, and reduction [9].
The Mutation Operator is used due to the rule for generating mutants from the original program.

Mutation Operator	Description
AAR	array reference for array reference replacement
ABS	absolute value insertion
ACR	array reference for constant replacement
AOR	arithmetic operator replacement
ASR	array reference for scalar variable replacement
CAR	constant for array reference replacement
CNR	comparable array name replacement
CRP	constant replacement
CSR	constant for scalar variable replacement
DER	DO statement alterations
DSA	DATA statement alterations
GLR	GOTO label replacement
LCR	logical connector replacement
ROR	relational operator replacement
RSR	RETURN statement replacement
SAN	statement analysis
SAR	scalar variable for array reference replacement
SCR	scalar for constant replacement
SDL	statement deletion
SRC	source constant replacement
SVR	scalar variable replacement
UOI	unary operator insertion

Figure 2: Operator of Mutation Analysis [8]

Fig. 2 is the example of the first set of Mutation Operator used in traditional mutation. Therefore, the next process is to test set ***T,*** which is supply towards the system. The test needs to be executed against the original program '***P'*** before starting mutation analysis to identify the correction of test cases. It needs to be fixed before running other mutants if *p* is incorrect. The mutant '***P'*** is labelled as killed if the result of '***P'*** is different from ***P*** in test case ***T***.

Technique proposed

Similarity Relation and Conditional Entropy: The construction of this new technique will be applied in mutation to reduce the large number of mutants

and to select the subset of operator mutants without reducing the effectiveness of the testing.

Similarity Relation removes redundancy of the mutants by classifying the same mutant in classes for reducing the number of mutants, while conditional entropy selects the mutant operator based on the classification of mutants. Based on previous work [17], the redundancy of mutants occurs due to the mutation operator generating various numbers of mutants and some mutation operators being able to generate more mutants.

Remove the data that have the redundancy, it is the safely reduction technique [17]. Further details of the Similarity Relation and Conditional Entropy steps are illustrated as following data set:

Data Set: The Data Set used in this research can be found online [16]. This data contains 85 mutants that obtained the MOs Mutants Operator. Fig. 3 shows the MOs Graphical User Interface (GUI).

Firstly, the data set is transformed into matrix table. The matrix table consists of column and row. The column lists the mutants, while the row lists the mutant operator.

The decision for each test is either kill or alive. Value '1' in the column and row indicates that the mutants are satisfied by a mutant operator, while value '0' means the mutants are unsatisfied by a mutant operator. Table 1 shows the mutants matrix table (MMT).

Table 1: Mutants Matrix Table

	REW	SWI	REL	AIW	ASW	ADW	MWS	EWWAR - RWWAR	RWHW	STATUS
M001	1	0	0	0	0	0	0	0	0	alive
M002	1	0	0	0	0	0	0	0	0	alive
M01	1	0	0	0	0	0	0	0	0	alive
M02	1	0	0	0	0	0	0	0	0	alive
M03	1	0	0	0	0	0	0	0	0	alive
M04	1	0	0	0	0	0	0	0	0	alive
M05	1	0	0	0	0	0	0	0	0	alive
M06	1	0	0	0	0	0	0	0	0	alive
M0	1	0	0	0	0	0	0	0	0	alive

7										
M08	1	0	0	0	0	1	0	0	0	alive
M09	1	0	0	0	0	0	0	0	0	alive
M010	1	0	0	0	0	0	0	0	0	alive
M011	1	0	0	0	0	0	0	0	0	alive
M012	0	1	0	0	0	0	0	0	0	alive
M013	0	1	0	0	0	0	0	0	0	alive
M014	0	1	0	0	0	0	0	0	0	alive
M015	0	1	0	0	0	0	0	0	0	alive
M016	0	1	0	0	0	0	0	0	0	alive
M017	0	1	0	0	0	0	0	0	0	alive
M018	0	1	0	0	0	0	0	0	0	alive
M019	0	1	0	0	0	0	0	0	0	alive
M020	0	1	0	0	0	0	0	0	0	alive
M021	0	1	0	0	0	0	0	0	0	alive
M022	0	1	0	0	0	0	0	0	0	alive
M023	0	0	1	0	0	0	0	0	0	alive
M024	0	0	1	0	0	0	0	0	0	alive
M025	0	0	1	0	0	0	0	0	0	alive
M026	0	0	1	0	0	0	0	0	0	alive
M027	0	0	1	0	0	0	0	0	0	alive
M028	0	0	1	0	0	0	0	0	0	alive
M029	0	0	1	0	0	0	0	0	0	alive
M030	0	0	0	1	0	0	0	0	0	alive
M0	0	0	1	0	0	0	0	0	0	alive

31										
M032	0	0	1	1	0	0	0	0	0	alive
M033	0	0	1	1	0	0	0	0	0	alive
M034	0	0	1	1	0	0	0	0	0	alive
M035	0	0	1	1	0	0	0	0	0	alive
M036	0	0	0	1	0	0	0	0	0	alive
M037	0	0	0	1	0	0	0	0	0	alive
M038	0	0	0	1	0	0	0	0	0	alive
M039	0	0	0	1	0	0	0	0	0	alive
M040	0	0	0	1	0	0	0	0	0	alive
M041	0	0	0	1	0	0	0	0	0	alive
M042	0	0	0	1	0	0	0	0	0	alive
M043	0	0	0	1	0	0	0	0	0	alive
M044	0	0	0	0	0	0	0	0	0	alive
M045	0	0	0	1	0	0	0	0	0	alive
M046	0	0	0	1	0	0	0	0	0	alive
M047	0	0	0	0	1	0	0	0	0	alive
M048	0	0	0	0	1	0	0	0	0	alive
M049	0	0	0	0	1	0	0	0	0	alive
M050	0	0	0	0	1	0	0	0	0	alive
M051	0	0	0	0	1	0	0	0	0	alive
M052	0	0	0	0	0	0	0	0	0	alive

M053	0	0	0	0	1	0	0	0	0	alive
M054	0	0	0	0	1	0	0	0	0	alive
M055	0	0	0	0	1	0	0	0	0	alive
M056	0	0	0	0	1	0	0	0	0	alive
M057	0	0	0	0	1	0	0	0	0	alive
M058	0	0	0	0	0	1	0	0	0	alive
M059	0	0	0	0	0	1	0	0	0	alive
M060	0	0	0	0	0	1	0	0	0	Alive
M061	0	0	0	0	0	1	0	0	0	alive
M062	0	0	0	0	0	1	0	0	0	Alive
M063	0	0	0	0	0	1	0	0	0	alive
M064	0	0	0	0	0	1	0	0	0	alive
M065	0	0	0	0	0	1	0	0	0	alive
M066	0	0	0	0	0	1	0	0	0	Alive

M068	0	0	0	0	0	1	0	0	0	alive
M069	0	0	0	0	0	0	1	0	0	alive
M070	0	0	0	0	0	0	1	0	0	alive
M071	0	0	0	0	0	0	1	0	0	alive
M072	0	0	0	0	0	0	1	0	0	alive
M073	0	0	0	0	0	0	1	0	0	alive
M074	0	0	0	0	0	0	1	0	0	alive
M075	0	0	0	0	0	0	1	0	0	alive
M076	0	0	0	0	0	0	1	0	0	alive
M077	0	0	0	0	0	0	1	0	0	alive
M078	0	0	0	0	0	0	1	0	0	alive
M079	0	0	0	0	0	0	0	1	0	alive
M080	0	0	0	0	0	0	0	1	0	alive
M081	0	0	0	0	0	0	0	0	1	alive
M082	0	0	0	0	0	0	0	0	1	alive
M083	0	0	0	0	0	0	0	0	1	Alive
M084	0	0	0	0	0	0	0	0	1	alive
M085	0	0	0	0	0	0	0	0	1	alive

TABLE I: MOs for GUI testing techniques

#	Class	Mutant Operator	Acronym
1	Rem.	–Remove Existing Widget	REW
2		–Set Widget Invisible	SWI
3		–Remove Existing Listener	REL
4	Adding	–Add Identical Widget	AIW
5		–Add Similar Widget	ASW
6		–Add Different Widget	ADW
7		–Add Another Listener	AAL
8	Modifying	–Expand/Reduce size of Windows and Widgets will Auto-adjust Their Sizes	EWWAR/ RWWAR
9		–Expand size of windows and widgets will not auto-adjust their sizes	EWWNAR/ RWWNAR
10		–Reduce size of windows to hide widgets	RWHW
11		–Modify location of a widget to a proper location	MLWP
12		Modify location of a widget to edges of windows	MLWE
13		–Modify location of a widget to overlap with another	MLWO
14		–Modify size of widgets	MWS
15		–Modify appearance of widgets	MWA
16		–Modify type of widgets (Button changed to TextField)	MWT
17		–Modify GUI library for widgets (Swing button changed to AWT Button)	MWL
18		–Expand/Reduce size of Windows and Widgets will Adjust their Sizes	EWWAR/ RWWAS

Figure 3: MOs GUI

Mutants classification using similarity relation

The process starts with the creation of Mutant Matrix Table (MMT), such as Table 1. Given a complete$MMT = (U, O, R, f)$, where $O = O *\cup \{d\}$, $M*$ is a set of condition attributes and d decision attribute, such that,$f: U \text{ X } M \rightarrow V$, for any $m \epsilon M$, where *V*, is called as the domain of attribute of m. In *MMT*, similarity relation, *S*, can be defined for any subset of B$\subseteq M *$, which is defined based on Definition 1 [18].

Definition 1*:* Let $MMT = (U, O, R, f)$, be a complete MMT. Similar class $I_B^S(M_m)$, of mutants with reference to Operator Mutant, *B*, is defined as $I_B^S(M_m) = \{M_n | M_n \epsilon U\}$.
Based on definition 1, similar classes can be easily obtained by analysing Table 1 with similarity relation.

Operator Mutants selection using conditional entropy

Conditional Entropy is used in this research to reduce a certain number of common Operator Mutants attributes. Based on definition 2 [18], conditional entropy can be calculated as follows:

Definition 2: $IIS = (U, C \cup \{d\})$ and B $\subset$ C is an incomplete information system. Let $U|I_B = \{I_B(X_1), I_B(X_2), \ldots, I_B(X_{|U|})\}$, $U \mid d = \{d_1, d_2, \ldots d_m\}$. The conditional entropy can be defined as following equation:

$$EN\ (d \mid B) = -\sum_{i=1}^{|U|} p(I_B(x_i)) \text{x} \sum_{j=1}^{|U|d|} p\left(d_j \middle|\ I_B(x_i)\right) \log_2 P\ (d_j | I_B(x_i))$$

Where

$$p(I_B(x_i)) = \frac{|(I_B(x_i)|}{|U|}, \qquad i = 1, 2, \ldots |U|$$

$$p(d_j | I_B(x_i) = \frac{p\,(I_B(x_i), d_j)}{p(I_B(x_i))} = \frac{|I_B(x_i) \cap d_j|}{|I_B(x_i)|} \qquad i = 1, 2, \ldots |U|\ j = 1, 2, \ldots m$$

Hence,

$$EN\,(d \mid B) = -\sum_{i=1}^{|U|} p(I_B(x_i)) \sum_{j=1}^{|U|d|} p\,(d_j | I_B(x_i))\ \log_2 P\,(d_j | I_B(x_i))$$

$$= -\sum_{j=1}^{|U|d|} \frac{|I_B(x_i)|}{|U|} \sum_{j=1}^{|U|d|} \frac{|I_B(x_i) \cap d_j|}{|I_B(x_i)|}\ \log_2 \frac{|I_B(x_i) \cap d_j|}{|I_B(x_i)|}$$

$$= \sum_{i=1}^{|U|} \sum_{j=1}^{|U|d|} \frac{|I_B(x_i) \cap d_j|}{|U||} \log_2 \frac{|I_B(x_i)\,|}{|I_B(x_i) \cap d_j|}$$

Reduction rate calculation

The evaluation of effectiveness of proposed technique based on the reduction rate. The following equation is used to calculate the rate of reduction.

Definition 3: Let *m(M*) be the number of reduced mutants, while *n(M)* be the number of actual mutants. The reduction rate of mutants can be calculated as follows:

$$= \frac{(n\,(M) - m\,(M))}{n\,(M)} * 100$$

RESULTS AND DISCUSSION

Similarity Relation and Conditional Entropy are implemented to improve the effectiveness of Mutant Selection. Firstly, classify the mutants in the same class. Secondly, the selection of mutant operator is demonstrated. Then, the rate of reduction is evaluated to verify the effectiveness of testing. From definition 1 in previous section, the similar classes can be presented as:

$$|S| = \left| \left\{ \begin{array}{l} \left\{ \begin{array}{l} mo-1, mo-2, m01, m02, m03, m04, m05, m06, m07, \\ m09, m010, m011 \end{array} \right\}, \\ \{m08\}, \left\{ \begin{array}{l} m012, m013, m014, m015, m016, m017, m018, m019, \\ m020, m021, m022 \end{array} \right\}, \\ \{m023, m024, m025, m026, m027, m028, m029, m031\}, \\ \{m030, m045, m046\}, \{m032, m033, m034, m035\}, \\ \{m036, m037, m038, m039, m040, m041, m042, m043\}, \\ \{m044, m052\}, \left\{ \begin{array}{l} m047.m048, m049, m050, m051, m053, \\ m054, m055, m056, m057 \end{array} \right\}, \\ \left\{ \begin{array}{l} m058, m059, m060, m061, m062, m063, m064, m065, \\ m066, m067, m068 \end{array} \right\}, \\ \left\{ \begin{array}{l} m069, m070, m071, m072, m073, m074, m075, m076, \\ m077, m078 \end{array} \right\}, \\ \{m079, m080\}, \{m081, m082, m083, m084, m085\} \end{array} \right\} \right| = 13$$

Conditional Entropy: From definition 2 in previous chapter, the calculation of conditional entropy is as below:

Step 1: Given a complete MMT in Table 1, we have:

$$O* = \{REW, SWI, REL, AIW, ASW, ADW, MWS, EWWAR - RWWAR, RWHW\}$$

$$\frac{U}{IND(d)} = \left\{ \begin{Bmatrix} m001, m002, m01, m02, m03, m04, m05, m06, m07, \\ m08, m09, m010, m011, m012, m013, m014, m015, \\ m016, m017, m018, m019, m020, m021, m022, \\ m023, m024, m025, m026, m027, m028, m029, m030, \\ m031, m032, m033, m034, m035, m036, m037, m038, \\ m039, m040, m041, m042, m043, m044, m045, m046, \\ m047, m048, m049, m050, m051, m052, m053, m054, \\ m055, m056, m057, m058, m059, m060, m061, m062, \\ m063, m064, m065, m066, m067, m068, m069, m070, \\ m071, m072, m073, m074, m075, m076, m077, m078, \\ m079, m080, m081, m082, m083, m084, m085 \end{Bmatrix} \right\}$$

$$S_M*(\boldsymbol{M001}) = \begin{Bmatrix} \{m001, m002, m01, m02, m03, m04, m05, m06, m07, \\ m09, m010, m011\} \\ =*(mo2) =*(m01) =*(m02) =*(m03) =*(m04) \\ =*(m05) =*(m06) =*(m07) =*(m09) =*(m010) \\ =*(m011), \end{Bmatrix}$$

$$S_M*(\boldsymbol{M012}) = \begin{Bmatrix} \{m012, m013, m014, m015, m016, m017, m018, m019, \\ m020, m021, m022\} \\ =*(m013) =*(m014) =*(m015) =*(m016) =*(m017) \\ =*(m018) =*(m019) =*(m020) =*(m021) =*(m022), \end{Bmatrix}$$

$$S_M*(\boldsymbol{M023}) = \begin{Bmatrix} \{m023, m024, m025, m026, m027, m028, m029, m031\} \\ =*(m024) =*(m025) =*(m026) =*(m027) \\ =*(m028) =*(m029) =*(m030) =*(m031), \end{Bmatrix}$$

$$S_M*(\boldsymbol{M036}) = \begin{Bmatrix} \{m036, m037, m038, m039, m040, m041, m042, m043\} \\ =*(m037) =*(m038) =*(m039) =*(m040) \\ =*(m041) =*(m042) =*(m043), \end{Bmatrix}$$

$$S_M*(\boldsymbol{M047}) = \begin{Bmatrix} \{m047, m048, m049, m050, m051, m053, m054, m055, \\ m056, m057\} \\ =*(m048) =*(m049) =*(m050) =*(m051) =*(m053) \\ =*(m054) =*(m055) =*(m056) =*(m057), \end{Bmatrix}$$

$$S_M*(\boldsymbol{M058}) = \begin{Bmatrix} \{m058, m059, m060, m061, m062, m063, m064, m065, \\ m066, m067, m068\} \\ =*(m058) =*(m059) =*(m060) =*(m061) =*(m062) \\ =*(m063) =*(m064) =*(m065) =*(m066) =*(m067) \\ =*(m068) \end{Bmatrix}$$

$$S_M * (\boldsymbol{M069})$$
$$= \begin{Bmatrix} \begin{Bmatrix} m069, m070, m071, m072, m073, m074, m075, m076, \\ m077, m078 \end{Bmatrix} \\ =*(m070) =*(m071) =*(m072) =*(m073) \\ =*(m074) =*(m075) =*(m076) * (m077) \\ =*(m078), \end{Bmatrix}$$
$$S_M * (\boldsymbol{M081}) = \begin{Bmatrix} \{m081, m082, m083, m084, m085, \} \\ =*(m082) =*(m083) =*(m084) =*(m085), \end{Bmatrix}$$
$$S_M * (\boldsymbol{M032}) = \{m032, m033, m034, m035\} =*(m033) =*(m034) =*(m035),$$
$$S_M * (\boldsymbol{M030}) = \{m030, m045, m046\} =*(m045) =*(m046)$$
$$S_M * (\boldsymbol{M044}) = \{m044, m052\} =*(m052),$$
$$S_M * (\boldsymbol{M079}) = \{m079, m080, \} =*(m080)$$
$$S_M * (\boldsymbol{M08}) = \{m08\}$$
$$EN(d \mid O*)$$
$$= -\begin{bmatrix} 12 * \frac{12}{85}(\frac{73}{12} log\, 2 \frac{73}{12}) + 11 * \frac{11}{85}\left(\frac{11}{11} log\, 2 \frac{11}{11}\right) + 8 * \frac{8}{85}\left(\frac{8}{8} log\, 2 \frac{8}{8}\right) \\ +10 * \left(\frac{10}{10} log\, 2 \frac{10}{10}\right) + 11 * \frac{11}{85}\left(\frac{11}{8} log\, 2 \frac{11}{8}\right) + 10 * \frac{10}{85}\left(\frac{10}{10} log\, 2 \frac{10}{10}\right) \\ +5 * \frac{5}{85}\left(\frac{5}{5} log\, 2 \frac{5}{5}\right) + 4 * \frac{4}{85}\left(\frac{4}{4} log\, 2 \frac{4}{4}\right) + 3 * \frac{3}{85}\left(\frac{3}{3} log\, 2 \frac{3}{3}\right) \\ * \frac{2}{85}\left(\frac{2}{2} log\, 2 \frac{2}{2}\right) + 2 * \frac{2}{85}\left(\frac{2}{2} log\, 2 \frac{2}{2}\right) + 1 * \frac{1}{85}\left(\frac{1}{1} log\, 2 \frac{1}{1}\right) \end{bmatrix}$$
$$= -26.8454$$

Step 2: When finished calculating the complete MMT, such as Step 1, we need to calculate the multiple mutant operators. The probability of getting the same value as Step 1 is analysed. These are some examples that have the same value as Step 1.

$$O* = \{REW + ADW\}$$
$$S_M * (\boldsymbol{M001}) = \begin{Bmatrix} m001, m002, m01, m02, m03, m04, m05, m06, \\ m07, m09, m010, m011 \end{Bmatrix}$$
$$S_M * (\boldsymbol{M08}) = \{m08\}$$
$$S_M * (\boldsymbol{M012}) = \begin{Bmatrix} m012, m013, m014, m015, m016, m017, m018, m019, \\ m020, m021, m022, m023, m024, m025, m026, m027, \\ m028, m029, m030, m031, m032, m033, m034, m035, \\ m036, m037, m038, m039, m040, m041, m042, m043, \\ m044, m045, m046, m047, m048, m049, m050, m051, \\ m052, m053, m054, m055, m056, m057, m069, m070, \\ m071, m072, m073, m074, m075, m076, m077, m078 \\ , m079, m080, m081, m082, m083, m084, m085 \end{Bmatrix}$$
$$S_M * (\boldsymbol{M058}) = \begin{Bmatrix} m058, m059, m060, m061, m062, m063, m064, m065, \\ m066, m067, m068 \end{Bmatrix}$$
$$EN(d \mid O*) = -\begin{bmatrix} 12 * \frac{12}{85}(\frac{73}{12} log\, 2 \frac{73}{12}) + 1 * \frac{1}{85}\left(\frac{1}{1} log\, 2 \frac{1}{1}\right) \\ +63 * \frac{63}{85}\left(\frac{63}{63} log\, 2 \frac{63}{63}\right) + 11 * \frac{11}{85}\left(\frac{11}{11} log\, 2 \frac{11}{11}\right) \end{bmatrix}$$
$$= -26.8454$$

Hence, this example shows the same values with Step 1. The Mutant Operators selected are REW and ADW. So, from nine (9) mutant operators, conditional entropy reduces it to two (2) mutant operators.

Step 3: When Steps 1 and 2 are completed, the reduction rate is calculated according to definition 3 in the previous chapter.

The results show that with similarity relation, the percentage of mutants reduced is 85%. From 85 mutants in the test set, only 13 mutants need to be executed against, while from Nine (9) mutant operators, only two (2) mutant operators remained. The fault detection capability is effective because all the data sets are tested to safely remove redundant data. The results are clearer in Table 2.

Table 2: Final Results

	REW	ADW
m001, m002, m01, m02, m03, m04 m05, m06,m07,m09,m010,m011	1	0
m012, m013, m014, m015, m016, m017, m018, m019,m020,m021,m022	0	0
m023, m024, m025, m026, m027, m028, m029, m031	0	0
m036, m037, m038, m039, m040, m041, m042, m043	0	0
m047, m048, m049, m050, m051, m053, m054, m055,m056,m057	0	0
m058, m059, m060, m061, m062, m063, m064, m065,m066,m067,m068	0	1
m069, m070, m071, m072, m073, m074, m075, m076,m077,m0078	0	0
m081, m082, m083, m084, m085	0	0
m032,m033,m034,m035	0	0
m030,m045,m046	0	0
m044,m052	0	0
m079,m080	0	0
m08	1	1

CONCLUSION

In conclusion, reducing the number of mutants and mutant operators for execution is necessary for successful mutation, but the percentage of effectiveness must not decrease. Therefore, the percentage of effectiveness of the quality of testing must also not diminish. Similarity Relation and Conditional Entropy are ideal methods for reducing the number of mutant and selecting mutant operator without randomly and these methods do not reduce the effectiveness of testing. Similarity Relation classifies the same class to ensure all mutants are analysed and executed without any discarded data. Conditional Entropy selects mutants based on formula to make sure the pattern of mutant and mutant operator is the same. Overall, the experiment of

Similarity Relation and Conditional Entropy is successful in contributing to the selection of optimum mutant without reducing the percentage of effectiveness testing. Moreover, using this technique, processing time for testing is saved and the cost due to the large number of mutants is reduced. Further research can be done to improve the method in mutation analysis.

ACKNOWLEDGMENTS

The researcher would like to thank University Tun Hussein Onn Malaysia (UTHM) and the Ministry of Higher Education Malaysia for supporting this research.

REFERENCES

R. Carvalho, A Comparative Study of GUI Testing Aproaches, Faculdade de Engenharia da Universidade do Porto, 2016.

F. M. Shaikh, S. Sabir, M. Abbas, An Optimized Approach for Graphical User Interface Testing, ResearchGate Publication, RawalPindi, 2015, pp 1-8.

D. Amalfitano, N. Amatucci, A. R. Fasolina, P. Tramontana , A Conceptual Framework for The Comparison of Fully Automated GUI Testing Technique, 2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), 2015, pp. 50-57.

R. S. Chhillar, A REVIEW: GUI TESTING, 2014.

R. M. L. M Moreira, A. C. Paiva, M. Nabuco, A. Memon, Pattern-Based GUI Testing: Bridging the gap between design and quality assurance, Software Testing, Verification and Reliability, vol. 27, no. 3, 2017, p.e1629.

T. Wetzlmaier, R. Ramler, Hybrid monkey testing: enhancing automated GUI tests with random test generation, Proceedings of the 8th ACM SIGSOFT International Workshop on Automated Software Testing, 2017, pp. 5-10.

S. Nedyalkova, J. Bernardino, Open Sourcw Capture and Replay Tools Comparison, Proceedings of the International C* Conference on Computer Science and Software Engineering, 2013, pp. 117-119.

Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE Transactions on Software Engineering, vol. 37, no. 5, 2010, pp. 649-678.

T. T. Chekam, Automated and Scalable Mutation Testing, 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST) , 2017, pp. 559-560.

J. Zhang, Z. Wang, L. Zhang, D. Hao, L. Zang, S. Cheng, L. Zhang, Predictive Mutation Testing, Proceedings of the 25th International Symposium on Software Testing and Analysis, Saarbrucken, Germany: ACM, 2016, pp. 342-353.

F. Wu, J. Nanayati, M. Harman, Y. Jia, J. Krinke, Memory Mutation Testing, Information and Software Technology, vol. 81, 2017, pp. 97-111.

S. J. Guillaume, Mutant Selection Using Machine Learning Techniques, en. In: Machine Learning: Theory and Applications, 2015, p. 24.

M. Shahid, S. Ibrahim, M. N. Mahrin, A Study on Test Coverage in Software Testingm, Advanced Informatics School (AIS), Universiti Teknologi Malaysia, International Campus, Jalan Semarak, Kuala Lumpur, Malaysia, 2011.

M. Linares-Vásquez, G. Bavota, M. Tufano, K. Moran, M. Di Penta, C. Vendome, C. Bernal-Cárdenas, D. Poshyvanyk, August. Enabling mutation testing for android apps, Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, 2017, pp. 233-244.

Y. Wei, MuDroid : Mutation Testing For Android Apps, Univ. College London, London, UK, Tech. Rep, 2015.

R. A. P Oliveria, E. Alegroth, Z. Gao, A. Memon, Definition and Evaluation of Mutation Operators for GUI-Level Mutation Analysis, International Conference On Software Testing, Verification and Validation Workshop, Windsor, UK: IEEE Computer Society, 2015, pp. 1-10.

P. Ammann, M. E. Delamaro, J. Offutt, Establishing Theoretical Minimal Sets of Mutants, 2014 IEEE International Conference on Software Testing, Verification and Validation, 2014, pp. 21-30.

N. F.M Nasir, Test Case Reduction Using Similarity Relations and Conditional Entropy, UTHM: Degree Master, 2018.

A. Dhankhar, S. Kamna, A Comprehensive Review of Tools & Techniques for Big Data Analytics in International Journal of Emerging Trends in Engineering Research, vol. 7, no.l 11, 2019, pp. 556–562.

S. Bhanu J, Baswaraj D., Bigul S. D., & JKR. Sastry, Generating test cases for testing embedded systems using combinatorial techniques and neural networks based learning model" in International Journal of Emerging Trends in Engineering Research, vol. 7, no. 11, 2019, pp. 417–429.