FAULT LOCALIZATION BY USING HYBRID GENETIC ALGORITHM

Muhammad Luqman Bin Mahamad Zakaria, Khaironi Yatim Sharif, Abdul Azim Abd. Ghani, Koh Tieng Wei, Hazura Zulzalil

Software Engineering and Information System Department,
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia, 43400 Serdang, Selangor, Malaysia

luqman.mzakaria@gmail.com
{khaironi, azim, twkoh, hazura}@upm.edu.my

Abstract

Maintaining software is difficult, time-consuming and very costly. Thus finding correct faults in a program might help to overcome this issue. Many techniques had been proposed such as program slicing, code coverage, program state and mutation analysis. An issue arise when all the previous technique are made based on the assumption that the fault is caused by a single fault. However, in a real world, one fault could be caused by multiple faults. Thus this issue requires a technique which able to handle multiple faults when to localize a fault. Another issue is related to the latest technique which is mutation analysis. During test cases mutation, there is a vast number of mutant will be generated which lead to difficulty of choosing the important mutant. Therefore there is a need for a new technique which can capable of reducing the number of mutants generates to localize the fault. HGA which is a combination of Genetic Algorithm and Local Search had been identified by another researcher as more efficient in searching optimum solution. Thus it is believed that HGA technique is suitable to localize fault in a source code. This paper will address the issue regarding a technique to localize a correct fault and handle multiple faults.

Keywords: Software maintenance, Fault localization, Hybrid Genetic algorithm.

1. Introduction

Nowadays, there is a various number of system had been developed by the programmer to facilitate human activities. A lot of effort had been put by the programmer during the development and testing to ensure that the system working well but the bug still exist in the system (W. Wong & Debroy, 2009). Finding and locating bug in a system is quite difficult. Even an experienced developer could make a wrong assumption when they want to locate the bugs (P. Alexandre, A. Rui, 2014). In order to fix the bugs or error in a source code, fault localization needs to be carried out. Fault localization is a process of finding the actual location of a fault. As a manual process is time-consuming, researchers had proposed a few technique which will help to find the actual fault location in an effective way with less time and less cost (W. Wong & Debroy, 2009).

Most of the previous proposed technique are made with the assumption that it is caused by a single fault. In a large and complex system, a bug could result from multiple bugs which can occur simultaneously (P. Alexandre, A. Rui, 2014). There is also possibilities that one fault can be caused by a combination of different fault (Rui Abreu, Zoeteweij, & van Gemund, 2008). Therefore, there is a need for a technique which capable of handle occurrences of multiple faults. The latest research also had focused on the use of mutation analysis in fault localization. They had proposed the used of mutation analysis because mutant produced by this technique can act as a real fault (Debroy & Wong,
2014). The result had shown that mutation analysis performs better compare to statement-based in term of fault localization. The problem with mutation analysis are there will be a lot of mutant or test cases will be created and how to choose the mutant that's lead to the root of the faults (Papadakis & Le Traon, 2014). Therefore, a new technique is required in order to localize the fault with a minimal number of test cases applied.

2. Fault Localization Technique

Fault localization can define as the process finding the actual fault or locate the exact location of a fault (Ali, Andrews, Dhandapani, & Wang, 2009; Cellier, Ducassé, Ferré, & Ridoux, 2008; Debroy & Wong, 2014; Ju et al., 2014). A lot of technique had been proposed by different researchers to improve fault localization. One of the technique is using program slicing. Program slicing is based on the idea of remove unwanted or unnecessary statement from a source which did not directly or indirectly effect the variables value in order to narrow down the search (P. Alexandre, A. Rui, 2014). Mark Weiser had introduced static program slicing in 1981 (Weiser, 1984). Program slicing can be divided into three type which is static and dynamic. One of the latest research using program slicing is Slice-based statistical fault localization (Mao, Lei, Dai, Qi, & Wang, 2014). The researcher had shown that their proposed technique are able to correlate program entities with the failures which are better than code coverage based technique.

Another technique category had been proposed are called Code coverage based. Code coverage based will use the execution information that had been recorded from the pass and failed test cases to locate the fault (P. Alexandre, A. Rui, 2014) (P. Alexandre, A. Rui, 2014). Spectrum-based fault localization is the technique under code coverage. It will use pass or fail execution information to trace the fault. Tarantula is one of spectrum based technique (Jones & Harrold, 2005). In this technique, suspiciousness of the code had been calculated and rank to localize the fault. Another researcher also had applied nearest neighbour query (Renieres & Reiss, 2003). The author had tried to overcome input limitation by proposed this technique. The problem with this technique is the author only consider one fault per program. Instead, focus on a single fault, a researcher had focused on handling multiple faults by using spectrum based reasoning (R. Abreu, Zoeteweij, & Gemund, 2009). In this method, the author had combine Bayesian reasoning with spectrum fault localization called Barinel. The latest review had shown that code coverage which using execution profile capable of handling multiple faults and can perform well, regardless of fault quantity, with only a small loss in effectiveness.

Program State is another technique category which based on the idea of simplifying the input for fault detection during debugging (P. Alexandre, A. Rui, 2014; W. Wong & Debroy, 2009). Failing test cases will be used as input for Delta debugging algorithm and this algorithm will remove input entity which will not cause any error until it reaches the minimum test cases. Another proposed technique are call predicate switching (Zhang, Gupta, & Gupta, 2006). In this paper, it will find the error output by comparing the actual output with the expected output. Then it will look for nearest predicate instance or conditional statement for switching purpose. After it is switching, then it will check the execution either it is success or not. Reasoning based is another technique which will use information from the system model and the runtime behaviour to detect the actual location of the fault (P. Alexandre, A. Rui, 2014; W. Wong & Debroy, 2009). System model will determine the behaviour and the behaviour will be observed and identified.
Recent research had focused on the use mutation analysis in improving fault localization. One of the paper is doing analysis about the effectiveness of mutation analysis in fault localization (Papadakis & Le Traon, 2014). In this paper, the author had claimed that mutant performs better compare to statement based. This paper tries to deal with the issue of a vast number of mutant by using selective mutation. The problem with the selective mutation are it will remove any operators which are almost similar in order to reduce the mutant but it did not consider the facts that different program classes require different sets of mutation operators (Adamopoulos, Harman, & Hierons, 2004). Another research also had proposed the use of mutation analysis for automate bug fixing and combine it with the fault localization (Debroy & Wong, 2014). An issue arises when it comes to choosing of mutants in this program. There is strategy needed from the programmer or software engineer when the proposed system wants to choose the mutant for fixing the fault.

3. Genetic Algorithm

Genetic algorithm is based on the idea that it will manipulate the population of the potential solution in order to optimize search or problem (Srinivas & Patnaik, 1994). This technique had been introduced in 1970's by Harrold and will involve three process which is selection, crossover and mutation. Genetic algorithm had been widely used in a different field. One of the applications of genetic algorithm under software maintenance are in source code refactoring (Ouni, Kessentini, Sahraoui, & Boukadoum, 2012). In this paper, the author had applied Genetic algorithm to choose an optimal solution for repairing defects in a source code. Different solution will be evaluated until the best and optimal solution is found. One of the paper had proposed the used of genetic algorithm by combining it with fault localization (Le Goues, Nguyen, Forrest, & Weimer, 2012). In this paper, genetic algorithm had been applied for choosing the suitable solution to fix the bug only but it is not applied in fault localization process. Genetic algorithm also had been used to identify the factor that affects the effectiveness of code coverage based technique (Masri, Abou-Assi, El-Ghali, & Al-Fatairi, 2009). Factors will act as the chromosomes and the crossover had been performing to check the effects of the factor on the technique.

Another paper had claimed that GA can be exploited to produce test cases automatically (Berndt, Fisher, Johnson, Pinguikar, & Watkins, 2003; Seesing, 2006). One application of SGA in breeding software test cases (Berndt et al., 2003). This paper is aim on improving test case generation to facilitate testing process by adapting GA selection function only. Triangle program had been used for test case data generation and it consists of x, y and z value for each individual population. In this paper, value for x, y and z will be initialized randomly in digit format. Fitness function of each chromosome will be compared with the previous test cases based on their novelty (uniqueness of test cases), proximity (closeness to other test cases) and severity (seriousness of a system error). Another researcher also had applied GA in test data generation (Pargas, Harrold, & Peck, 1999). This paper aims to generate program input based on the testing requirement. In this method, test cases are chosen to be the initial population. The fitness function is calculate based on the predicates in CDGPaths. For crossover process, any individual that higher predicate number will be selected to be a parent for next test case generation. In this method, one-point crossover had been choose. Iteration of population generation will stop when the target test cases are found or it had reach max attempt.

GA also had been apply in test case prioritization (Mohapatra, 2013). Test case prioritization is aim to reduce test case selection in order to reduce the time taken for a testing process by following code coverage, execution time and severity of the test case. Input for GA technique proposed by the author
are program and list of test cases, number of initial population, number of test case to be taken excluding severe, maximum iterations, crossover probability, mutation probability and initial Population. Test cases had been select as an initial population.

In fault localization, GA had been used to improve ranking technique based on the execution profiles (CHEN, 2011). Four type execution profiles had been used which is number of failed test cases, number of passed test cases, number of failed test cases that executed the profile entity and number of passed test cases that executed the profile entity. In this research, each individual or chromosomes are represented in the form of a tree that contains set of math operators and set of different execution profile variables. Genetic operators apply for a new generation of the population are replication, population and crossover. GA is used to select ranking function that has highest fitness value. New population will be generate based on the mathematical operators and the execution profile data. Another paper had discussed GA technique in fault detection (Ouni et al., 2012). In defects detection, GA will be used to get the best set of rules. A chromosome consists of tree diagram which will link between a set of metric with the defects type. In crossover, crossover operator will allow the features (sub tree) of the best fitted parent to be transmitted to a randomly choose parent. Drawbacks of SGA technique are there is no memorization, optimization is not linear, risk of suboptimal solution and delayed convergence. There also possibility that the optimal solution is not achieved from the SGA (Mala, Ruby, & Mohan, 2010).


Our proposed solution aims to improve software fault localization by letting multiple faults can be localized simultaneously. Our proposed framework consist of 2 phase. Each phase will be describe in next section. Figure 1 shows the flow of proposed framework. Test cases are required to for method selection. The method that has suspiciousness value greater than zero will be pass to next phase for code modification. This phase is aim to check the impact of the fault in one method towards another method. If the suspiciousness value of other method had been reduced, that the fault is found.

![Figure 1: HGA Fault Localization framework](image-url)
4.1. Method Selection

The process involves in method selection is shown in Figure 2. Method selection requires a set of test cases to be execute throughout the source code in order to find the path execution of the class and method (CM). Each class and method will be recording their execution profiles. Execution profiles are the collected program execution information that will be used to rank suspiciousness (CHEN, 2011). Execution profiles consist of information such as the number of failed test cases that execute statement (Failed (S)), number of passed test cases that execute statement (Passed (S)) and number of failed test cases (totfailed). This information will be collected during runtime execution.

![Method Selection Process](image)

Once the execution profile had been recorded, suspiciousness for each method and class will be calculated. In order to calculate the suspiciousness for each executes method, Ochiai Suspiciousness formula had been choose (Equation 3.1). Selection of Ochiai suspiciousness is based on the claim of other paper that Ochiai Suspiciousness formula is more effective compared to Tarantula or Jaccard (R. Abreu et al., 2009; Assiri & Bieman, 2014; Santelices, Jones, Yu, & Harrold, 2009; W. E. Wong & Debroy, 2009). Once the suspiciousness value for each method had been calculated, all the method will be arranged according to their suspiciousness value. Those method that have suspiciousness value more than zero will be select to the next phase.

\[
suspiciousness(s) = \frac{failed(s)}{\sqrt{totfailed \times (failed(s) + passed(s))}}
\]

![Ochiai Suspiciousness Formula](image)
4.2. Code Modification by using Hybrid Genetic Algorithm (HGA)

Hybrid Genetic algorithm is a population-based approach which combine genetic algorithm and local search technique to optimize searching problem (Land, 1998). One of the research had applied HGA to improve test case quality by calculating mutation score and path coverage (Mala et al., 2010). HGA had been choose because it more efficient compares to GA. The step involves in HGA is shown in Figure 4. Hybrid genetic algorithm is used for source code modification. To perform this operation, HGA requires an initial population to be defined. In this proposed framework, the initial population is choosing from the set of fault category. Each category will represent chromosomes. Each chromosome contain a set of operators such as Arithmetic Operator Replacement (AOR), Relational Operator Replacement (ROR), Conditional Operator Replacement (COR), Assignment Operator Replacement (AOR), Statement Deletion, Remove any part of the source code, Replacement of Boolean, Absolute Value Insertion and Wrong input (Ma, Offutt, & Kwon, 2006).

![Figure 4: HGA step in code modification](image)

Once the initial population had been initialized, RemoveTop will be execute. RemoveTop will eliminates chromosome that will not help in generate a new set of source code. In this case, unnecessary fault category will be remove first. Next step will be LocalOpt. LocalOpt will remove operators that have lowest fitness function. Fitness function that will be use are based on the suspiciousness value. Generated source code will be calculated their suspiciousness and their impact of modification towards other method call. If the suspiciousness value is highest the current value, the operators will be eliminates. Crossover and mutation will takes place in order to choose operators that will be used to generate a set of source code which has lowest suspiciousness value. Each operator applied to the source code will be repeated evaluate. The stopping condition for the HGA cycle is zero suspiciousness value.
5. Experiment:

An experiment will be conducted in order to validate our proposed tool. The experiment will be conducted with a large system and a small system. Performance measurement, benchmark tools, and dataset will be explained in detail in this section.

5.1. Variables and Measures

5.1.1. Independent Variables

Fault localization tools had been chosen as a dependent variable for this experiment. Three fault localization tools had been selected which are Barinel, Tarantula, and Ochiai. Barinel is a program that combines abstract program traces and the Bayesian reasoning to deduce multiple fault candidates. Ochiai and Tarantula had been selected because both of the methods had been claimed that they are effective in fault localization (R. Abreu et al., 2009; Assiri & Bieman, 2014; Santelices et al., 2009; W. E. Wong, Debroy, Gao, & Li, 2014).

5.1.2. Dependent Variables

To compare the performance of our tools, one independent variable had been chosen which is effectiveness. To measure the effectiveness, the best EXAM and worst EXAM score will be calculated for each tool (Jones & Harrold, 2005). A tool is considered effective if the EXAM score is higher. In order to measure the EXAM Score, each statement in the dataset will be ranked. The ranking is based on the suspiciousness value.

5.2. Performance Measurement

Different datasets had been used to test performance. Most of the researcher had used the Siemens dataset to evaluate the performance of the proposed technique (Rui Abreu, Mayer, Stumptner, & van Gemund, 2009; Rui Abreu, Zoeteweij, & Van Gemund, 2009; Eric Wong, Debroy, & Choi, 2010; Masri et al., 2009). Siemens dataset consists of seven programs which are print_tokens, print_tokens2, replace, schedule, schedule2, tcas, tot_info. Another paper also had used Unix dataset for evaluation (Eric Wong et al., 2010; Mao et al., 2014; Zhang et al., 2006). In our experiment, 7c Program will be used to measure effectiveness is small scale system. For large scale system, GZIP, UNIX, space, and grep will be selected.

6. Analysis

To conduct the analysis, the statistical model that will be used is Analysis of variance (ANOVA). ANOVA will be used because there are three tools that will be compared with the proposed framework in the experiment. Indeed, this experiment will only measure the effectiveness only.
7. Conclusion

This research is expected to produce a new framework for fault localization in a source code. The proposed framework should be capable of identifying the part of source code that contains the root cause of the fault, determined correct fault from generating mutants, and capable of handling single and multiple faults in a source code.

Acknowledgement

This paper is funded by Ministry of Education (MOE) Malaysia under the Fundamental Research Grant Scheme (FRGS) (08-01-16-1850FR).

References


