Automated Test Case Generation for Effective Spectrum-based Fault Localization

Bo Yang, Yuze He and Lixing Feng
North China University of Technology, Beijing, China
yangbo090313@163.com

Abstract. Software testing is the key to ensuring software quality. Test case generation and software fault localization are two important research objects of testing. Spectrum-based fault localization (SBFL) method is a mainstream dynamic fault localization method, which inputs test cases into the program for execution and collects statement coverage information and execution results. By analysing this information, a fault report can be generated. The performance of SBFL is affected by test case quality. Therefore, this paper proposes a test case generation method based on improved genetic algorithm to assist SBFL. This method uses a small number of initial test cases to execute the program, then uses the XGBoost model to pre-process the spectrum information and generates a list of potential fault statements to guide the iterative process of the genetic algorithm. The experimental results show that the test cases generated using the improved genetic algorithm have certain advantages over the test cases generated by general genetic algorithm and the baseline test cases.

1. Introduction
Test case generation and software fault location are the research focuses of software testing. The purpose of fault localization is to find faults in the program by analyzing the static or dynamic characteristics of the program. Program spectrum refers to the set of statement coverage information at runtime, it is the description of the program’s dynamic behaviour. The running of SBFL requires the support of test cases. During the execution, test cases are only valid if they covered the faulty statements. Therefore, it is important to choose a reasonable strategy to generate test cases.

Genetic algorithm is a typical heuristic algorithm that simulates the process of natural evolution. Jones et al. [1] combined genetic algorithm with structured test case generation for the first time. On this basis, there are many researches on test case generation using genetic algorithm. For example, Shen et al [2] proposed an automatic generation method of software test data based on genetic algorithm and tabu search algorithm. This method has better local search capability and global search capability. Pinto et al [3] proposed a multi-objective framework for test case generation. NSGA-II (Non-Dominant Sorting Genetic Algorithm) is used to evaluate and optimize the solutions of different objectives.

In addition to genetic algorithm, there are many test case generation methods based on other heuristic algorithms, such as ant colony algorithm, particle swarm algorithm, simulated annealing algorithm, etc. At present, there is little research on the test case generation based on heuristic algorithm for software fault localization, especially for SBFL.

This paper proposes a test cases generation method for the SBFL using an improved genetic algorithm. XGBoost [4] is used to analysis the spectrum information when program is executed with a
small number of initial test cases to generate a potential faulty statement list. A fitness function that can use the parameter of potentially faulty statements is constructed to guide the evolution process. The contributions of this paper are as follows:

1. Improve the genetic algorithm by using dynamic crossover and mutation operator according to the variety of the population and combine it with test case generation.
2. Use the XGBoost model to analyse the spectrum information with a small number of test cases to find the potential faulty statements. Then use them to guide the generation of test cases.
3. It can generate high-quality test cases with high statement coverage, condition coverage, and fault statement coverage for the spectrum-based fault localization method.

The remainder of this paper is as follows: Section 2 is the background of SBFL, XGBoost and genetic algorithm. Section 3 describes the method proposed in this paper. Section 4 introduces the experiment carried out and section 5 summarizes the paper.

2. Background

2.1. Spectrum-based fault localization
SBFL is a dynamic fault localization method that analyzes the coverage information of the program statements to find suspicious statements. When test case $t$ is input into program $P$, if the execution result meets the expected output, then $t$ is a successful test case, i.e. $t \in t_s$. Otherwise, $t$ is considered to be a failed test case, i.e. $t \in t_f$. SBFL analyzes the coverage of program statements in these two test cases through a specific model to rank the suspiciousness of the statements.

The implementation of SBFL is based on some certain assumptions, mainly including:

**Assumption 1**: The faulty statements in the program may be covered by successful test cases or failed test cases when the program is executed (The incidental correctness of faults).

**Assumption 2**: A failed test case will cover at least one faulty statement when executed.

Wong et al. [5] represent the coverage information of the program executed by the test cases in the form of a matrix. As shown in figure 1, suppose program $P$ contains a total of $N$ statements and there are one or more faults. At the same time, test suite $T$ has $M$ test cases. These $M$ test cases contain a certain number of successful test cases $t_s$ and failed test cases $t_f$. Each row element $s$ in the coverage matrix represents the number of times each statement of the program $P$ is executed while executing a certain test case. The $r$ corresponding to each row represents the result of the test case (0 represents the successful test case, 1 represents the failed test case). This paper uses the coverage matrix and result vector to represent program spectrum information.

\[
\begin{bmatrix}
    s_1 & s_2 & s_3 & \cdots & s_N \\
    s_1' & s_2' & s_3' & \cdots & s_N' \\
    1 & 1 & 1 & \cdots & 1 \\
    1 & 1 & 1 & \cdots & 1 \\
    1 & 1 & 1 & \cdots & 1
\end{bmatrix}
\]

**Figure 1.** Coverage matrix and result vector.

2.2. XGBoost
XGBoost is an algorithm ensemble learning algorithm in the field of machine learning proposed by Chen et al. [4]. Ensemble learning builds and combines multiple learners to complete learning tasks. XGBoost has a wide range of applications and excels in classification, regression, and ranking prediction problems on structured data.

XGBoost uses decision rules to classify and calculate the scores of each leaf to obtain the final prediction result. The process of tree construction of XGBoost model is a process of continuously selecting of splitting point, the more times a feature is selected for splitting, the more important it is.
For our problem, by transforming the program spectrum information into a coverage matrix and input it into the XGBoost model, each column in the coverage matrix is a feature, which means that each feature is corresponding to a statement in the program. If a feature is selected multiple times for splitting, it means that it has a great impact on classification, that is, the corresponding statement has a great impact on the results of program’s execution. In other words, it is the potential faulty statement.

2.3. Genetic Algorithm

The genetic algorithm treats the potential solutions of the problem as "individuals" (for this paper, test cases) and encodes them as chromosomes. Then, the fitness function is used to evaluate the individual according to the corresponding requirements, and new individuals are generated through the three operations of selection, crossover, and mutation.

Selection operator is used to select individuals from the population with a certain probability, so that they have the opportunity to crossover and pass the excellent genes to the next generation. This paper adopts the roulette wheel selection as the selection operator. The roulette wheel selection operator first calculates the fitness of each individual, and then calculates the proportion of this fitness in the total fitness of the population.

Crossover operator is used to select a pair of good individuals as parents based on the crossover probability and pass the excellent genes to the next generation through genetic recombination. This paper uses two-points crossover as crossover operator. The crossover operator selects a pair of individual’s genes from the population according to the crossover probability, and then selects the equal-length gene fragments from the genes of the two individuals and exchanges them, thereby generating two new individuals.

Mutation operator selects an individual from the population according to the mutation probability, and then selects one or more gene points on its chromosome for mutation operation. This paper chooses simple mutation as mutation operator.

3. Proposed method

3.1. Improved genetic algorithm

As mentioned in section 2.1, only test case that covers faulty statements is valid. Therefore, this paper improves the genetic algorithm by adding the position parameter of potential faulty statements generated by the XGBoost model to the fitness function, so as to improve the coverage of faulty statements. To do so, first use a small number of initial test cases to run the program and collect spectrum information. The spectrum information is converted into the coverage matrix and result vector and input into the XGBoost model. The accuracy of classification is adjusted by 10-fold cross-validation. After the classification, a list of potential faulty statement is generated and used to guide the generation of test cases.

In addition, this paper also designs an adaptive crossover and mutation operator to dynamically adjust the crossover probability and mutation probability by measuring the variety of the population, thereby improving the overall search efficiency.

3.2. Construction of fitness function

The fitness function is used to evaluate the scores of individuals in the population. The fitness function in this paper is constructed with three weighted parameters: the score based on the distance between the covered statement and the potentially faulty statement, the statement coverage, and the number of covered conditions.

Score based on the distance between covered statement and potential faulty statement

After the program under test is executed by the initial test case, the XGBoost model classifies and ranks the importance of the features, and returns a list of potential faulty statements. Then the weight \(w\) is calculated for each potential faulty statement according to the length of the list. For an individual \(i\), the score can be calculated by:
\[dscore = \sum_{i=1}^{n} \frac{1}{1 + \min(d_i)} w_i\] (1)

Where \(d_i\) is the distance between the covered statement and the \(i\)-th potential faulty statement. \(n\) is the length of potential faulty statement list.

**Statement coverage**

Statement coverage refers to the percentage of statements covered in the program when the test case is executed.

**Number of covered conditions**

A condition is a component of a branch statement in a program, and its possible results are true or false. The number of covered conditions refer to the number of conditions that the test case can cover after the execution of program.

**Fitness function**

The fitness function is a weighted combination of the above three parameters. For an individual \(s_i\) in the population, the fitness can be calculated by the following formula:

\[fit(s_i) = \alpha dscore(s_i) + \beta coverage(s_i) + \gamma condition(s_i)\] (2)

### 3.3. Adaptive genetic operator

During the evolution of the population, it is necessary to ensure the variety of the population. In order to reduce the precocity and prevent the variety from decreasing, the improved genetic algorithm proposed in this paper uses dynamic crossover and mutation probability according to the variety of the population.

**Population variety**

Variety is a measure of the breadth of the gene’s distribution in a population. This paper uses the variance of individual’s fitness \(s\) to calculate the variety of the population. Population variety can be calculated by:

\[V = \frac{1}{N} \sum_{i=1}^{N} (fit_i - fit_{avg})^2\] (3)

Where \(fit_i\) is the fitness of individual \(s_i\) in the population, \(fit_{avg}\) is the average fitness of all individuals in the current population.

**Adaptive crossover operator**

The crossover probability \(P_c\) is an important parameter in the genetic algorithm and has a great influence on the evolution of the population. This paper uses a dynamic crossover probability to increase or maintain the crossover probability in a round of iteration based on the variety of current generation and previous generation. The crossover probability can be calculated by:

\[Pc_i = \begin{cases} Pc_{basic} & V_i \geq V_{i-1} \\ Pc_{basic} + Pc_{variance} \cdot \text{sigmoid}(\frac{V_{i-1}}{V_i}) & V_i < V_{i-1} \end{cases}\] (4)

\(Pc_{basic}\) is a preset basic crossover probability. \(Pc_{variance}\) is a preset amount of change in probability. The Sigmoid function is used to normalize the ratio of Varieties. \(V_i\) is the variety of individuals in the current population, and \(V_{i-1}\) is the variety of individuals in the previous generation.

**Adaptive mutation operator**

Similar to the crossover operator, this paper uses the dynamic mutation probability to increase or maintain the mutation probability in current round of iteration based on the variety of the previous generation. The mutation probability can be calculated by:
\[ P_{m} = \begin{cases} P_{m_{\text{basic}}} & V_i \geq V_{i-1} \\ P_{m_{\text{basic}}} + P_{m_{\text{variance}}} \cdot \text{sigmoid}(\frac{V_{i-1}}{V_i}) & V_i < V_{i-1} \end{cases} \] (5)

\( P_{m_{\text{basic}}} \) is a preset basic crossover probability. \( P_{m_{\text{variance}}} \) is a preset amount of change in probability. The Sigmoid function is used to normalize the ratio of Varieties. \( V_i \) is the variety of individuals in the current population, and \( V_{i-1} \) is the variety of individuals in the previous generation.

4. Experiment

In order to verify the effectiveness of the proposed improved genetic algorithm IGA for generating test cases for SBFL technology, this paper selects four programs from the Siemens suite in the SIR library and conducts an experiment. The test cases generated by IGA are compared with the test cases provided by the SIR library and the test cases generated by the general GA.

4.1. Object program

The Siemens suite is one of the most commonly used evaluation benchmarks in the field of software testing. This paper selects 4 programs from Siemens suite, as shown in table 1. Each program in the Siemens Suite has a fault-free version and several faulty versions, each faulty version containing one or more manually implanted faults. At the same time, the Siemens suite provides test cases for each of these programs to facilitate dynamic methods like SBFL.

| Program   | Number of testcase | Highest coverage | Average coverage | Average number of covered conditions | Faulty statement coverage |
|-----------|--------------------|------------------|------------------|-------------------------------------|--------------------------|
| Print_token | 4130              | 31.44%           | 18.65%           | 38.60                               | 43.82%                   |
| Print_token2 | 4115             | 36.08%           | 25.31%           | 61.28                               | 76.30%                   |
| Tot_info  | 1052              | 26.60%           | 18.79%           | 46.16                               | 86.53%                   |
| Tcas      | 1608              | 31.21%           | 24.04%           | 10.53                               | 7.65%                    |

4.2. Experiment setup

For the IGA proposed in this paper, the test cases in the experiment are in the form of files and are encoded in Unicode. The selection operator is roulette wheel selection, and the crossover operator is two-point crossover. In the setting of crossover probability, \( P_{c_{\text{basic}}} \) is 0.6 and \( P_{c_{\text{variance}}} \) is 0.2. The mutation operator is single-point mutation. \( P_{m_{\text{basic}}} \) is 0.03 and \( P_{m_{\text{variance}}} \) is 0.01. In the fitness function, \( \alpha = 0.45, \beta = 0.35, \) and \( \gamma = 0.2 \) For the general GA, a fixed crossover and mutation probability is used. crossover probability \( P_c \) is 0.7 and a mutation probability \( P_m \) is 0.04. In the fitness function, \( \alpha = 0 \) (does not use potential faulty statements), \( \beta = 0.6, \) and \( \gamma = 0.4. \)

30 randomly selected test cases from the Siemens suite are used as initial test cases. During the population evolution process, the maximum number of individuals is set to 100 in order to generate more test cases based on the initial test cases. Both genetic algorithms iterate 30 times and independently run 10 times to take the average of the results.

4.3. Experiment result

Table 2 shows the coverage information of the test cases generated by the IGA and the GA iterated for 15 and 30 rounds (The form of data in the table is: the result of 15 iterations (the result of 30 iterations)). It can be seen that in Highest coverage and Average coverage, in most cases, when the number of iterations is the same, the performance of IGA has some advantages over GA. In Average number of covered conditions and Faulty statement coverage, the advantages of IGA are obvious. For example, in Print_token1, Tot_info and Tcas, when IGA and GA both iterate for 15 rounds, the faulty statement coverage of the test cases generated by IGA is 10% higher than GA.
The test cases generated by IGA have great advantages over the test cases provided by the Siemens suite. In all four programs, the highest coverage of test cases generated by IGA after 30 iterations is greater than or equal to the test cases in the Siemens suite. The Average coverage of test cases generated by IGA are 12.88%, 11.31%, 11.35%, and 4.04% higher than those of the Siemens suite. On the Average number of covered conditions, the test cases generated by IGA are 40.77, 49.37, 30.8, and 4.34 higher than those of the Siemens suite. On the Faulty statement coverage, the test cases generated by IGA were 56.18%, 23.7%, 13.47%, and 80.5% higher than those of the Siemens suite.

### Table 2. Coverage information of generated testcase

| Program   | Method | Highest coverage | Average coverage | Average number of covered conditions | Faulty statement coverage |
|-----------|--------|------------------|------------------|--------------------------------------|--------------------------|
| Print_token | IGA    | 31.08% (31.97%)  | 26.66% (31.53%)  | 62.82 (79.37)                        | 95.20% (100%)            |
| GA        |        | 30.37% (31.97%)  | 25.87% (30.06%)  | 58.04 (73.18)                        | 85.50% (100%)            |
| Print_token2 | IGA    | 36.67% (36.87%)  | 34.60% (36.62%)  | 101.89 (110.65)                      | 100% (100%)              |
| GA        |        | 36.08% (36.87%)  | 32.53% (35.89%)  | 91.28 (106.46)                       | 100% (100%)              |
| Tot_info  | IGA    | 30.37% (30.73%)  | 26.54% (30.14%)  | 62.50 (76.96)                        | 93.56% (100%)            |
| GA        |        | 30.19% (30.73%)  | 23.92% (28.93%)  | 54.02 (70.44)                        | 85.46% (100%)            |
| Tcas      | IGA    | 31.21% (31.21%)  | 28.08% (28.90%)  | 13.56 (14.84)                        | 62.60% (87.80%)          |
| GA        |        | 31.21% (31.21%)  | 27.10% (28.67%)  | 13.01 (13.78)                        | 53.10% (83.00%)          |

### 5. Conclusion

This paper proposes an improved genetic algorithm for SBFL to generate high quality testcase. It uses the XGBoost model to calculate the location of potential faulty statements and construct a fitness function to utilize this information. It also uses the adaptive crossover and mutation operators to dynamically adjust the crossover probability and mutation probability. The experiment results show that the test cases generated by the improved genetic algorithm IGA proposed in this paper are compared with test cases generated by common genetic algorithms and benchmark test cases. Test cases generated based on the IGA algorithm have certain advantages in highest coverage, average coverage, average number of covered conditions, and faulty statement coverage. Therefore, we can say that the test case generation method based on the IGA algorithm is an effective method.

### Acknowledgement

This work was supported in part by NSFC (61502011, 61672041) Talent Special (107051360019XN132/025), Basic Research Business Fee (110052971803/038) Plan, Project of Beijing College Students’ Scientific Deep Research (108052961917/002) and Project of Dominant Discipline Construction in North China University of Technology (2020).

### References

[1] Jones, B. F., Sthamer, H. H., & Eyres, D. E. (1996). Automatic structural testing using genetic algorithms. *Software engineering journal, 11*(5), 299-306.

[2] Shen, X., Wang, Q., Wang, P., & Zhou, B. (2009, August). Automatic generation of test case based on GATS algorithm. In *2009 IEEE International Conference on Granular Computing* (pp. 496-500). IEEE.

[3] Pinto, G. H., & Vergilio, S. R. (2010, October). A multi-objective genetic algorithm to test data generation. In *2010 22nd IEEE International Conference on Tools with Artificial Intelligence* (Vol. 1, pp. 129-134). IEEE.

[4] Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794).

[5] Wong, W. E., & Qi, Y. (2009). BP neural network-based effective fault localization. *International Journal of Software Engineering and Knowledge Engineering, 19*(04), 573-597.