CIT. Journal of Computing and Information Technology, Vol. 28, No. 2, June 2020, 101–109
doi: 10.20532/cit.2020.1005134

Fault Localization Based on Hybrid Genetic Simulated Annealing Algorithm

Zhihua Zhang[1] and Yongmin Mu[2]

[1]Computer School, Beijing Information Science and Technology University, Beijing, China
[2]Beijing Key Laboratory of Internet Culture and Digital Dissemination Research, Beijing, China

Software testing is an important stage in the software development process, which is the key to ensure software quality and improve software reliability. Software fault localization is the most important part of software testing. In this paper, the fault localization problem is modeled as a combinatorial optimization problem, using the function call path as a starting point. A heuristic search algorithm based on hybrid genetic simulated annealing algorithm is used to locate software defects. Experimental results show that the fault localization method, which combines genetic algorithm, simulated annealing algorithm and function correlation analysis method, has a good effect on single fault localization and multi-fault localization. It greatly reduces the requirement of test case coverage and the burden of the testers, and improves the effect of fault localization.

ACM CCS (2012) Classification: Software and its engineering → Software creation and management → Software verification and validation → Software defect analysis

Keywords: hybrid genetic simulated annealing algorithm, function call path, fault localization

1. Introduction

Software testing is an important process in software engineering. It is the final inspection of products in each stage of software development before software release, and a process of detecting software faults and fixing them in order to ensure correctness, completeness, and consistency of software products. Software fault localization technology is an analysis method to locate the defects when some test cases fail to execute after the execution of test case sets [1]. Effective fault localization method can shorten the development cycle and improve the efficiency of error localization. Traditional software testing mainly focuses on manual testing, such as adding breakpoints to the program for single-step debugging, which is helpful for testers to be more familiar with the system and business processes, but it is time-consuming and labor-intensive and cannot guarantee the adequacy of testing. Traditional testing plays a very important role in the initial product testing, guarantees the quality of software to a certain extent and proves the importance of software testing. However, with the continuous iteration of products, the number of development iterations increases and testing becomes more and more complex. Manual testing gradually exposes many shortcomings and limitations. Especially when there is a high requirement for real-time and concurrency, manual testing is far from reaching the test goal. Therefore, traditional testing modes and methods must be reformed to improve testing efficiency. Automated testing technology emerges at the right moment [2, 3]. Compared with traditional manual testing, automated testing improves testing efficiency, reliability, and adequacy. Automation and semi-automation can greatly avoid the disadvantages brought by manual debugging, so the research of automatic fault localization has become one of the most popular research directions in software testing.

Existing genetic algorithms are affected by their poor local search ability when solving

search problems, which leads to inefficiency in practical application, and the difference between the final solution and the optimal solution is too large. In addition, most of the existing path-based defect location methods are based on the code level. For practical applications, there is often a large amount of code, and the number of paths obtained at the code level increases explosively [4, 5]. To solve these problems, a fault localization method based on hybrid genetic simulated annealing algorithm is proposed in this paper. It transforms fault localization into a software engineering problem based on search. This method uses the correlation degree between functions in the function call path as a penalty factor of fitness function of hybrid genetic simulated annealing algorithm, constructing the initial population and fitness function required by the algorithm. The location of faults is obtained from the results of the algorithm.

2. Concepts

2.1. Terminology Definitions

Definition 1. Function Call Graph. The call relationship between functions can be expressed as a directed graph $G = <V, R>$. V is a set of nodes, and each node represents a function. $R = \{(x, y) \mid x, y \in V\}$ is a set of arcs in the digraph, representing the call relationship or sequential execution relationship between functions. The arc $e = (T(e), H(e))$ connects two adjacent nodes $T(e)$ and $H(e)$ in G. $T(e)$ is the arc head and $H(e)$ is the arc tail.

Definition 2. Function Call Path. Function call path describes the execution path of the program source code in terms of function as a basic unit [6, 7]. W is a sequence of nodes $W_i = (V_1, V_2, \ldots, V_m)$. Adjacent nodes in a path represent call relationships or sequential execution relationships.

Definition 3. Test Suite $T = \{t_1, t_2, \ldots, t_m\}$ represents the test case set. t_i represents the i^{th} test case of the test case set.

Definition 4. Coverage Matrix $M = (M_{ij})$ denotes the coverage relationship between T and W. M is a matrix of $m*n$, and line i represents the function coverage of the i^{th} test case and the j^{th} column shows the coverage of the j^{th} function by different test cases. Each M_{ij} represents the coverage of the j^{th} function by the i^{th} test case. $M_{ij} = 1$ indicates that the test case i covers function j, and $M_{ij} = 0$ indicates that the test case i does not cover function j.

Definition 5. Failure Test Case Coverage Matrix $M_F = (M_{ij})$ is part of the coverage matrix M, which contains only coverage information for failed test cases.

Definition 6. Successful Test Case Coverage Matrix $M_p = (M_{ij})$ is part of the coverage matrix M, which contains only coverage information for successful test cases.

Definition 7. Result Vector $R = \{r_1, r_2, \ldots, r_m\}$ represents the test results of test cases. r_i represents the execution result of the i^{th} test case. $r_i = 0$ means that the i^{th} test case succeeded in the execution; $r_i = 1$ means that the i^{th} test case failed in the execution.

Definition 8. Adjacent Functions. Adjacent functions are represented by arcs in directed graph $G = <V, R>$ of function call relationship. Formal description: Two adjacent nodes $T(e)$ and $H(e)$ connected by arc $e = (T(e), H(e))$ in G are called adjacent functions.

Definition 9. Correlation Degree. The number of common paths of adjacent functions represents the correlation degree C_{rr} between functions. Formal description: $C_{rr} = e \in \{W_1, W_2, \ldots, W_m\}$, W_i represents the effective path in digraph G, that is, the function call path. C_{rr} denotes the number of paths containing e.

Definition 10. Evaluation Index of Multi-fault Localization. The multi-fault localization evaluation index returns the percentage of the code that needs to be checked to detect all faults.

2.2. Hybrid Genetic Simulated Annealing Algorithm

The Hybrid Genetic Simulated Annealing (HGSA) algorithm incorporates the idea of local search algorithm into the traditional genetic algorithm, strengthens the local search ability of genetic algorithms and further improves optimization quality and search efficiency, so as to make up for the shortcomings of the single opti-

mization method. In order to prevent premature puberty, HGSA preserves the diversity of the population while ensuring high adaptability of the population [9, 10].

There are many ways to combine GA (Genetic Algorithm) with SA (Simulated Annealing). This paper adopts an integrated idea. This idea takes advantage of the characteristics of SA heuristic search algorithm, combining SA heuristic search algorithm with a genetic algorithm, so as to change the algorithmic structure of GA, which improves the local search ability of genetic algorithm while retaining its global search ability. The optimum ideas are as follows:

Step 1: Initialization. Generating an initial feasible solution population with high quality and diversity by appropriate strategies.

Step 2: Using annealing temperature to control the number of iterations and termination conditions.

Step 3: Adding simulated annealing operation to selection operator and using metropolis criterion to select individuals retained to the next generation.

Step 4: Perform genetic operations, such as crossover, mutation, *etc.*

Step 5: Repeat Step 2, Step 3 and Step 4 until the termination condition of the algorithm is satisfied.

3. Proposed Approach

The entire process of our approach can be considered in three phases: the population initialization and fitness function design phase combined with the function call path, hybrid genetic simulated annealing algorithm based candidate population distribution heuristic search phase, and fault localization phase according to optimal population. Among the three phases, the main tasks of the first phase are chromosome encoding, population construction, fitness function design, and function relationship analysis. All the tasks in phase one are used to configure the input for the next phase; the second phase task models the fault localization problem and finds an optimal population, while retrieval of the initial population and fitness function design have been completed in the first phase. Thus, the second phase will focus on the design of a genetic operator, and on the temperature schedule; the last phase will map the optimal population obtained from the previous phase to concrete functions, so that the results can assist developers to localize faults.

3.1. Population Initialization and Fitness Function Design

3.1.1. Chromosome Encoding

According to the fault localization problem, we only need to know whether a function contains faults. Therefore, we design a binary vector to represent candidate fault distribution:

$$C = \{c_1, c_2, c_3, ..., c_n\}.$$

Here, n is the number of functions in the tested program, c_j shows whether there is a fault in function c_j, and if $c_j = 1$, then function c_j is considered to be faulty. If $c_j = 0$, then function c_j is considered to be free of faults in the candidate fault distribution.

3.1.2. Population Construction

In order to obtain a population constructed by multiple individuals, we first convert the execution path to chromosome codes. Based on the failure mechanism, we propose the following assumptions: (1) succeeded path may contain faults; (2) failed path must contain faults; (3) the common part of success path and failure path is a relatively low possibility to contain faults; (4) functions that exist only on failure paths have the highest possibility to contain faults.

Based on the above assumptions, we design a Total-Greedy (TG) algorithm to construct the initial population.

Algorithm 1. Total-Greedy (TG) algorithm.

Input: coverage matrix M, result vector R, number of individuals in population p N_p
Output: Initial population P

1. create the failure matrix M_{np}
2. create the diagonal matrix M_t
3. **while** scan M do
4. **if** the value of R is 1 && path include function then
5. add the vector to M_{np}
6. **end if**
7. **end while**
8. create the Map < Integer, Integer > *total*
9. **while** scan M_{np} do
10. *total* ← count sum of each column in the matrix M_{np}
11. **end while**
12. *length* ← number of failure cases
13. **while** scan M_t do
14. *distribution* ← an individual of M_t
15. *fitness* ← the ability to interpret a failure case
16. **while** *fitness* < *length* do
17. change distribution according to *total*
18. *fitness* ← the ability to interpret a failure case
19. **end while**
20. **end while**
21. $P = M_t$
22. return P

TG is an algorithm which can guarantee diversity of the population, while not hurting the quality of individuals. The algorithm first extracts failure coverage matrix M_{np} from coverage matrix M and result vector R, and calculates the occurrences (assigned to "total") of failure functions according to M_{np}, then generates a diagonal matrix representing n individuals, where each individual (candidate fault distribution) contains only one faulty statement, that is, there will be only one "1" element in the binary vector, while other elements are all "0". According to the diagonal matrix, the algorithm estimates whether each individual is able to explain all the failed test cases. If one individual fails to explain, it will be fixed based on "total", using roulette strategy to choose the function position, until an individual C is able to explain all the failure test cases. The N_p individuals generated by the TG algorithm have good quality and diversity, because each individual is extended from different fault locations. The high quality initial population will benefit from the convergence speed of the hybrid genetic simulated annealing algorithm.

3.1.3. Correlation Between Functions

In a program, a function is not isolated, but is associated with function call paths to implement specific tasks. Therefore, functions on the same function call path are related to each other. We quantify the relationship to correlation base on the following assumptions:

Assumption 1. Adjacent functions have the highest correlation, while the correlation between non-adjacent functions can be neglected.

Assumption 2. Correlation between two adjacent functions is proportional to the number of function call paths containing both of the functions.

Under the above assumptions, adjacent functions refer to functions adjacent to each other in one function call path, therefore, one function may have multiple adjacent functions, since it may occur in multiple function call paths.

3.1.4. Fitness Function

Similarity coefficients like Tarantula [13] and Ochiai [14] can detect suspiciousness of the program. However, functions in a program are not independent, they will affect each other, which will make the results detected from suspiciousness quite different from actual results. Thus, we have designed a Pro-Ochiai formula using the correlation between functions as a penalty function.

Pro-Ochiai suspiciousness formula is extended from Multi-Ochiai [14], introducing a penalty function to measure the suspiciousness of candidate fault distribution. Consider a candidate fault distribution C: its suspiciousness can be calculated by the following formulas.

$$\text{Pro-Ochiai}(C) = \frac{\varnothing(C)}{\sqrt{\left|T_F \times \left(\varnothing(C) + P(C) + R(C)\right)\right|}} \quad (1)$$

where $\varnothing(C)$ is the ability of C to explain failure test cases. The definition of $\varnothing(C)$ is

$$\varnothing(C) = \sum_{M_i \in M_F, 1 \le i \le |T_F|} \rho\left(\sum_{j=1}^{n} C_j \times M_{ij}\right), \quad (2)$$

where $\rho(x)$ is defined as:

$$\rho(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \end{cases}. \quad (3)$$

$P(C)$ is defined as:

$$P(C) = \sum_{M_i \in M_P, 1 \le i \le |T_P|} \sum_{j=1}^{n} C_j \times M_{ij}. \quad (4)$$

In the formulas above, M_{ij} is an element in Definition 4 Coverage Matrix, $|T_F|$ is the number of failed test cases, and $\varnothing(C)$ is the ability of candidate fault distribution C to explain failure test cases. If all the functions that C considers containing faults exist on the failure execution paths, we believe that C is able to explain failure test cases. On the contrary, if none of the functions on the failure execution path exists in C, then we think C is not able to explain that failure test case.

In formula (4), $|T_P|$ is the number of succeeded test cases in the test case set. $P(C)$ and $R(C)$ are two penalty functions. $P(C)$ is the number of test cases that C can explain. If a function in C is executed by a succeeded test case, then $P(C)$ increases by one. $R(C)$ is another penalty function which is designed to deal with the propagation effect of faults. If a candidate fault distribution determines that the number of faulty functions is greater than 2, then we need to check how many combinations of adjacent functions there are. If we assume that there are m combinations, then we obtain m correlations and sum them up, and the sum is $R(C)$. Otherwise, if N_f is less than 2, $R(C)$ will be 0. Benefiting from these two penalty factors, we are able to greatly enhance the efficiency of fault localization. Given a fault distribution, the higher its Pro-Ochiai, the better its effect of fault localization.

3.2. Optimization of HGSA

Hybrid Genetic Simulating Annealing (HGSA) algorithm starts with a subset of feasible candidate solutions (initial population), utilizing fitness function to keep the superior while dropping the inferior ones. It evolves to generate new individuals and to make sure that higher quality ones survive to the next generation. In each generation, HGSA makes selections, crossovers, and mutations according to fitness function and Metropolis criterion, in order to generate new individuals. HGSA utilizes the decrease of temperature to control the iteration until reaching the stopping temperature, and the final population will be the near-optimal solution to the problem.

The algorithm uses the population generated by TG as the initial population and Pro-Ochiai as the fitness function. The algorithm of HGSA is described as below.

Algorithm 2. HGSA algorithm.

Input: coverage matrix M, result vector R, initial temperature T

Output: near-optimal solution population P

1. $pop \leftarrow \text{TG}(M, R, N_p)$
2. $CandidatePool \leftarrow pop$
3. **Repeat**
4. $T \leftarrow T^*$ (1-coolingRate)
5. $C_{selected} \leftarrow \text{select}(pop, roulettewheel, GGAP)$
6. $C_{crossover} \leftarrow \text{crossover}(C_{crossover}, Crossover\_Rate, SA())$
7. $C_{mutated} \leftarrow \Phi$
8. **for** all $C \in C_{crossover}$ do
9. **if** $\text{mutate}(C) \le P_m$ then
10. $C_{mutated} \leftarrow \text{mutate}(C)$
11. **end if**
12. **end for**
13. $\text{recombine}(pop, C_{mutated})$
14. $CandidatePool \leftarrow CandidatePool \cup C_{mutated}$
15. until $T \le T_{end}$
16. $P \leftarrow \text{map}(CandidatePool)$
17. **return** P

The selection operator should let the elite individuals survive to the next generation or let their children survive to the next generation. The parameter GGAP ($0 < \text{GGAP} \le 1$) is used to determine the proportion of chosen individuals in a population. Roulettewheel refers to the roulette operator we use, which is similar to the casino game.

The crossover operator should combine parts of the parents to generate a new individual. HGSA uses shuffle crossover operator, which can improve diversity of the population. Concrete operations of the crossover operator are: from one of the parents select an entry with a certain probability at the same coordinate, and assign it to the child. Parameter Crossover_Rate is the possibility, and parameter SA() refers to the simulated annealing after the crossover.

The mutation operator should introduce changes to the children generated by the crossover operator, in order to speed the convergence utilizing mutation operator's ability of local random search when the solution has been close to the optimal solution due to genetic algorithm. Obviously, the function granularity in function call path and the aggregation effect of faults make parameter P_m smaller. The ordinary mutation operator used in this paper should decrease the number of mutations and guarantee the population quality.

3.3. Fault Localization

After obtaining the optimal candidate faults distribution, we need to convert the population to suspiciousness ranking list of functions. Obviously, the higher the suspiciousness of a function in the candidate faults distribution, the higher the function's possibility of obtaining faults. If there is only one function in the candidate faults distribution with the highest suspiciousness, then we consider that this function contains faults; if there are multiple functions in the candidate faults distribution with the highest suspiciousness, we need to make a ranking list of functions according to other candidate faults distributions with high suspiciousness.

4. Experiment and Evaluation

4.1. Evaluation Data Set

This paper uses the programs in Siemens Suites and GNU Software as experimental data. All these programs are from the SIR library. Siemens Suites is a small-scale program, and GNU Software is a large-scale program. Siemens Suites consists of seven programs, including print_tokens, print_tokens2, replace, schedule, schedule2, tcas, and tot_info. Each set of programs has multiple different defect versions and one correct version. Each version contains a manually implanted defect. There are many programs in GNU Software, and we have selected two programs: gzip and grep.

Experimental parameters are set in Table 1, where T_i represents the initial temperature during the simulated annealing algorithm, T_e represents the termination temperature of the simulated annealing process, coolingRate represents the decay rate of the temperature, and GGAP represents the proportion of the selected individual in each iteration to the candidate to be selected. P_c indicates the probability that the genes at the same position on two individual chromosomes will cross, and P_m represents the probability of chromosomal variation.

Table 1. Parameter settings in the experiment.

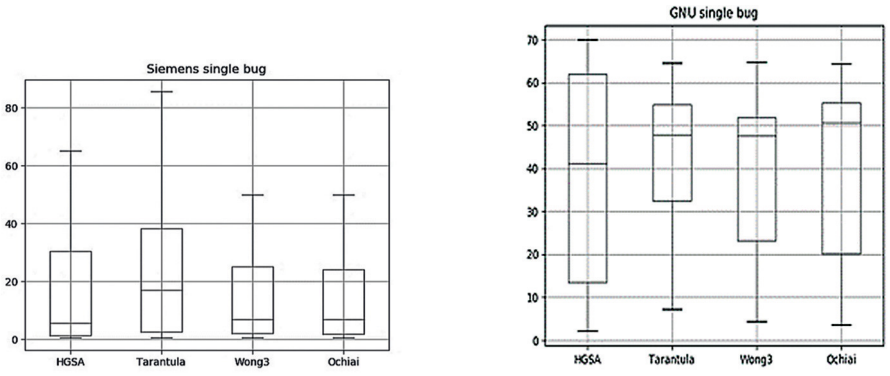
T_i	T_e	coolingRate	GGAP	P_c	P_m
1000	1	0.997	0.4	0.7	0.001

4.2. Experiment and Evaluation

4.2.1. Empirical Evaluation of Single Defect Location

This paper uses several classic defect localization methods to compare with HGSA, These defect localization methods are Tarantula, Wong3 [15] and Ochiai, and the target programs are Siemens Suites and GNU Software. The experiment mainly evaluates effectiveness of the algorithm from the positioning effect of a single defect version, and the standard adopted is EXAM. It refers to the percentage of program entities that need to be reviewed when finding the wrong program entity as a percentage of all program entities.

The experimental results are shown in Figures 1(a) and (b), and the data for Tarantula, Wong3 and Ochiai are quoted from literature. The ordinate in the figure indicates the percentage of all functions to be examined for finding the defect function, and the abscissa indicates the corresponding fault localization method. As shown



(a) *(b)*

Figure 1. Comparison of single defect positioning results.

in Figure 1(a), HGSA has similar defect positioning effects to Wong3 and Ochiai and is superior to Tarantula. Figure 1 (b) shows that the average positioning effect of HGSA is better. Figure 1 (a) and (b) show the defect positioning effect of each method from the box row diagram. Comparing the results of the two methods, HGSA's defect positioning effect is not inferior to the other three methods.

4.2.2. Multi-Fault Localization Empirical Evaluation

In the experiments with multiple faults localization, this paper selects multiple defect versions of four multi-fault Siemens programs (print_tokens, print_tokens2, replace, and tot_info) to evaluate multi-fault location problems. In the experiment, only the executable statements are considered. For each faulty version of the target program, run the corresponding algorithm 30 times and calculate the average, then get a list of corresponding suspicious functions. Finally, check each function according to the suspiciousness from high to low. Unlike single fault localization, the following metrics are used to evaluate the positioning of the method:

1. use the multi-fault localization standard of Definition 10, which effectively evaluates the effectiveness of multi-fault localization;
2. use the metrics of the percentage of the correct diagnosis, fitting the effect curve of fault localization and visually showing the effect of each fault localization method.

The traditional EXAM standard is used to evaluate experimental results. Table 2 shows the overall positioning effect of each algorithm applied to various incorrect versions of Siemens Suites. Each data R% in the table indicates the percentage of the total number of source code the method needs to review when locating the number of defects in R%. The first column indicates the percentage of the number of reviewed source codes. In this paper, the values are divided into ten intervals, and the proportion of faults that can be located in each interval is obtained. This data is an important indicator for measuring the effect of defect location.

Table 2. Method effect table.

Check code percentage	HGSA	Tarantula	Wong3	Ochiai
1%	45.18	13.93	29.55	36.36
1% ~ 10%	72.72	55.73	63.64	70.45
10% ~ 20%	77.24	61.47	65.91	75.00
20% ~ 30%	86.36	71.31	77.27	81.81
30% ~ 40%	90.91	79.51	79.55	86.36
40% ~ 50%	95.45	86.89	81.82	90.91
50% ~ 60%	98	87.71	86.36	95.45
60% ~ 70%	100	88.53	90.91	95.45
70% ~ 80%	100	92.63	93.18	98
80% ~ 90%	100	100	100	100
90% ~ 100%	100	100	100	100

Figure 2 is a line chart of the data in Table 2, where the abscissa indicates percentage of the statement that needs to be searched for the total statement; the ordinate indicates the proportion of faults we found in all faults. Each point (x, y) indicates that the fault localization method locates y% of all faults and needs to check x% of all statements. The smaller the x corresponding to the same y, the better the fault localization method. It can be clearly seen from the figure that the effect of HGSA is superior to the other three methods, although the amplitude is not large.

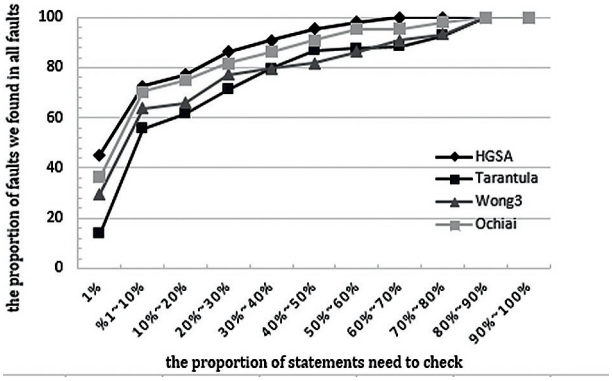


Figure 2. Fault localization effect under the EXAM standard.

In summary, the following conclusions can be drawn: the fault localization method based on hybrid genetic simulated annealing algorithm proposed in this paper has high accuracy. Whether for a single fault or multiple faults, the algorithm shows high accuracy and stability.

5. Conclusion

Research on software fault localization is a hot issue in the domain of software testing. This paper applies the heuristic search technology to the software testing, proposes a fault localization method based on hybrid genetic simulated annealing algorithm. On the one hand, the correlation matrix is constructed using the degree of correlation of functions between function calls and is applied to the fitness function of the hybrid genetic simulated annealing algorithm, which greatly improves the effectiveness of the hybrid genetic simulated annealing algorithm. On the other hand, granularity of the statement is raised to the function level, which reduces manual labour and material resources required for the whole testing process and improves the fault localization efficiency of the algorithm. In this paper, by designing an appropriate initial population, fitness function, genetic operator and simulated annealing process, the final suspiciousness ranking list of functions is obtained to assist developers in locating the faults. Finally, the experimental design of single fault and multiple faults is used to demonstrate the accuracy and effectiveness of the proposed fault localization method.

Acknowledgement

This work is supported by the Opening Project of Beijing Key Laboratory of Internet Culture and Digital Dissemination Research (5221935409), Beijing Municipal Natural Science Foundation (Z160002), and Graduate Education Funding (71D1811013).

References

[1] Z. Lei *et al.*, "Execution-Aware Fault Localization Based on the Control Flow Analysis", in *Proc. of the Information Computing and Applications (ICICA '10), Lecture Notes in Computer Science*, 2010, pp.158–165. http://dx.doi.org/10.1007/978-3-642-16167-4_21

[2] W. E. Wong *et al.*, "Effective Software Fault Localization Using an RBF Neural Network", *IEEE Transactions on Reliability*, vol. 61, no. 1, pp. 149–169, 2012. http://dx.doi.org/10.1109/TR.2011.2172031

[3] Y. Qi *et al.*, "Using Automated Program Repair for Evaluating the Effectiveness of Fault Localization Techniques", in *Proc. of the 2013 International Symposium on Software Testing and Analysis (ISSTA'13)*, 2013, pp.191–201. http://dx.doi.org /10.1145/2483760.2483785

[4] W. Wanzhi *et al.*, "A Technique of Multiple Fault Localization Based on Conditioned Execution Slicing Spectrum", *Journal of Computer Research and development*, vol. 50, no. 5, pp. 1030–1043, 2013.

[5] J. Xiaolin *et al.*, "Advances in Fault Localization Techniques", *Journal of Frontiers of Computer Science and Technology*, vol. 6, no. 6, pp. 481–494, 2012. http://dx.chinadoi.cn/10.3778/j.issn.1673-9418.2012.06.001

[6] L. Danfeng, "Research on Defect Location Method Based on FCP Path Slice", Master Dissertation, Beijing Information Science and Technology University, pp. 18–22, 2016.

[7] L. Danfeng and M. Yongmin, "Research of Fault Location Based on Function Calling Path and Faults Correlation Analysis", *Application Research of Computers*, vol. 33, no. 8, pp. 2363–2370, 2016. http://dx.chinadoi.cn/10.3969/j.issn.1001-3695.2016.08.028

[8] L. Jingyu and Z. Fengyu, "Research on Codes Defect Localization Rules by Calculating Defect Contribution Rate", *Application Research of Computers*, vol. 32, no. 9, pp. 2702–2707, 2015. http://dx.chinadoi.cn/10.3969/j.issn.1001-3695.2015.09.033

[9] X. Chen *et al.*, "Review of Dynamic Fault Localization Approaches Based on Program Spetrum", *Journal of Software*, vol. 26, no. 2, pp. 390–412, 2015. http://www.jos.org.cn/1000-9825/4708.htm

[10] W. U. Xiao-Ping *et al.*, "Research of Software Defect Prediction Model Based on LASSO-SVM", *Application Research of Computers*, vol. 30, no. 9, pp. 2748–2751, 2013. http://dx.chinadoi.cn/10.3969/j.issn.1001-3695.2013.09.047

[11] F. C. Meng *et al.*, "Solving SaaS Components Optimization Placement Problem with Hybrid Genetic and Simulated Annealing Algorithm", *Journal of Software*, vol. 27, no. 4, pp. 916–932, 2016. http://www.cnki.net/kcms/doi/10.13328/j.cnki.jos.004965.html

[12] J. A. Jones *et al.*, "Visualization of Test Information to Assist Fault Localization", in *Proc. of the IEEE Int. Conference on Software Engineering (ICSE'02)*, 2002, pp. 467–477. http://dx.doi.org/10.1145/581396.581397

[13] R. Abreu *et al.*, "On the Accuracy of Spectrum-based Fault Localization", in *Proc. of the Testing: Academic and Industrial Conference Practice and Research Techniques-Mutation (TAICPART-MUTATION'07)*, 2007, pp. 89–98. http://dx.doi.org/10.1109 /TAIC.PART.2007.13

[14] W. Zan *et al.*, "Genetic Algorithm Based Multiple Faults Localization Technique", *Journal of Software*, vol. 27, no. 4, pp. 879–900, 2016. http://www.cnki.net/kcms/doi/10.13328/j.cnki.jos.004970.html

[15] W. E. Wong *et al.*, "Effective Fault Localization using Code Coverage", in *Proc. of the 31st Annual International Computer Software and Applications Conference (COMPSAC '07)*, 2007, pp. 449–456. http://dx.doi.org/10.1109/COMPSAC.2007. 109

Received: November 2020
Revised: January 2021
Accepted: March 2021

Contact addresses:
Zhihua Zhang
Beijing Information Science and Technology University
Beijing
China
e-mail: zhang_zh@bistu.edu.cn

Yongmin Mu
Beijing Key Laboratory of Internet Culture and Digital Dissemination Research
Beijing
China
e-mail: yongminmu@163.com

Zhihua Zhang received the BSc degree and the MSc in computer science from the Harbin University of Science and Technology in 1993 and 1996, respectively. She is currently an associate professor of computer science and software engineering with Beijing Information Science and Technology University, China. She has long been engaged in software engineering teaching, and participated in related research work. Her research interests include software testing, software quality assurance and programming languages.

Yongmin Mu received the PhD degree in computer and automation from the China University of Mining and Technology in 1997. His research interests include software testing technology and test automation. Currently, he is a professor from the School of Computer, Beijing Information Science and Technology University, China. He is currently the director of the Open laboratory. He has long been engaged in architecture design, development and testing method of large scale software system. He has undertaken many national-level provincial and ministerial research projects.