Simultaneous Localization of Multiple Defects in Software Testing Based on Reinforcement Learning

Jiajuan Fang[1(✉)] and Yanjing Lu[2]

[1] Department of Software Engineering, Zhengzhou Technical College, Zhengzhou 450121, China

[2] Zhengzhou Technical College, Zhengzhou, China

Abstract. At present, most software defect localization methods focus on single defect localization, but few on multi-defect localization. Therefore, the multi-defect localization method based on reinforcement learning is proposed. By using genetic algorithm, the candidate distribution population can be transformed into a sort of suspicious value of real program entity, and the location of multiple defects in software testing can be realized simultaneously. Experimental results show that, compared with the average evaluation index of the existing methods, the evaluation index $EXAM_F$ of the proposed method is reduced by 1.19 and $EXAM_L$ reduced by 1.05, which shows that the proposed method has better positioning performance and is suitable for popularization.

Keywords: Reinforcement learning · Software testing · Multiple defects · Positioning · Genetic algorithm

1 Introduction

With the development of computer industry, software has become an indispensable part of modern information society. In order to meet the application requirements of various industries, the scale and complexity of software are increasing [1]. Due to complex requirements, imperfect project management, unreasonable development methods, imperfect development tools, irregular coding and so on, the number of bugs in software is increasing, and accidents are happening constantly. For example, in 1963, an American exploration rocket bound for Mars exploded because of a programmer's error. In the 1991 Gulf War, a Patriot missile killed 28 American soldiers stationed in Saudi Arabia because of a faulty system clock; In 1996, an Ariana rocket exploded due to a software flaw, causing great losses to the country and society; In 2000, a century-old computer software system crashed because of a design flaw, leading to costly software upgrades for all industries. In 2011, a design flaw in signalling equipment led to the July 23 high-speed train crash in Wenzhou. Although software provides people with a better quality of life, the economic loss, social loss and image loss caused by software defects are immeasurable.

W. Fu et al. (Eds.): ICMTEL 2021, LNICST 387, pp. 180–190, 2021.
https://doi.org/10.1007/978-3-030-82562-1_16

In order to guarantee the quality of software, a lot of testing is needed. But if the use case fails, the developer needs to check the code and find the defects. According to a 2002 report by the National Institute of Standards and Technology (NIST), software defects cost the US economy about $59.5 billion (0.6% of GDP) a year, and more than half of the cost of fixing or responding to them is borne by software users, with software developers and vendors paying the rest. In addition, for large software, large defect data makes direct manual inspection difficult. For example, Firefox receives up to 2.5 million software crashes a day, making it extremely difficult to locate defects manually. Therefore, how to use the existing information to quickly locate the defects in the software has become an important issue for researchers.

When the software fails, it needs to locate the software defects, which is one of the most expensive activities in the software development process. Finding an efficient software defect location technology is an important research topic. Testing is an important step in the software development process. Software defect location technology needs to use the information generated from testing for software defect location analysis. At present, most of the software defect location methods mainly focus on the problem of internal defect location under the assumption of single defect, but not enough on the unknown number of defects and system testing. In this paper, making a deep research on these problems, and propose a multi-defect simultaneous localization method for software testing based on reinforcement learning. Augmented learning is one of the paradigms and methodologies of machine learning, which is used to describe and solve the problem of agent maximizing reward or achieving specific goals through learning strategies in interaction with the environment. It hope to enhance the efficiency of bug location in software testing by enhancing the application of learning. It is innovation lies in the introduction of genetic algorithm, which transforms the candidate distribution population into a suspicious value of real program entity, and realizes the simultaneous localization of multiple defects in software testing, so as to enhance the superiority of genetic algorithm and improve the application performance of evaluation index.

2 Research on Simultaneous Location of Multiple Defects in Software Testing

2.1 Simultaneous Location of Multiple Defects Architecture

In order to meet the needs of today's social software, design multi-defect localization architecture for software testing based on reinforcement learning.

Reinforcement learning consists of mobile networks and evaluation networks. Mobile networks are the best thing you can do to your environment at the next moment, based on your current state [2]. For mobile networks, the reinforcement learning algorithm allows its output nodes to be randomly searched. With internal reinforcement signals from the assessment network, the output nodes of the mobile network can efficiently perform random searches and greatly improve the likelihood of selecting good actions, while training the entire mobile network online [3]. Using an auxiliary network to model the environment, the assessment network uses external reinforcement signals for predicting scalar values based on the current state and simulation environment, so that

it can predict one-step and multi-step action reinforcement signals currently imposed on the environment by the action network, and can provide advance reinforcement signals to the action network on candidate actions, as well as more rewarding and discouraging information (internal reinforcement signals) to reduce uncertainty and increase learning speed. The reinforcement learning structure is shown in Fig. 1.

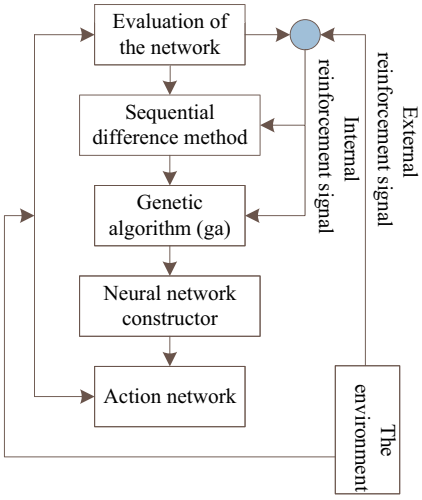

Fig. 1. Diagram of reinforcement learning structure

Based on Fig. 1, strengthen learning to build multiple defects in software testing and locate the architecture simultaneously, as shown in Fig. 2.

As shown in Fig. 2, multi-defect localization in software testing is divided into two stages. The first stage uses genetic algorithms to initialize the multi-defect population; then performs selection, crossover, and mutation operators to generate new individuals and add them to the population. At the same time, the Multi-Ochiai doubtful degree coefficient is used as the fitness value to evaluate the individuals and evolve into a new population. If the termination conditions are met, the final optimal multi-defect population will be obtained. Then enter the second stage, according to the optimal distribution of multiple defects in the program to get the corresponding entity suspicious ranking, the end of the algorithm [4].

2.2 Positioning Basic Block Partition

A basic block is a sequence of statements executed sequentially in a program, in which there is only one entry and one exit. In a method of defect location based on spectrum information, statements in the same basic block have the same coverage information, that is, they have the same degree of suspicion, and in the list of doubts, statements in the same basic block cannot distinguish the priority of the statements being checked [5]. Due to space limitations, this study shows only part of the code, as shown in Table 1.

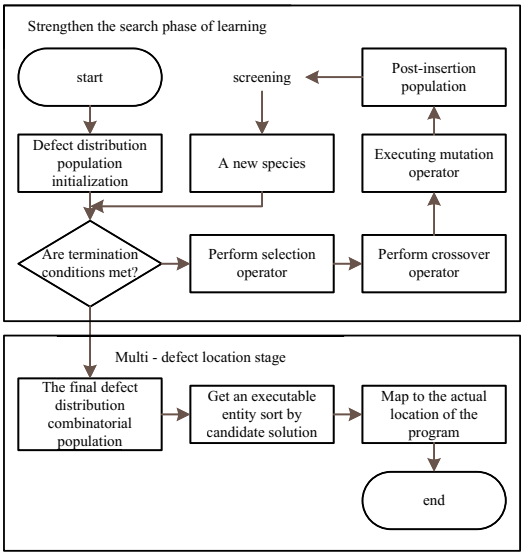

Fig. 2. Simultaneous localization architecture diagram for multiple defects in software testing

Table 1. Code part suspicious degree calculation result table

Code line number	Tarantula	Ochiai	Tarantula*
13	0.667	0.707	0.667
14	0.545	0.375	0.205
15	0.545	0.375	0.205
16	0.769	0.625	0.481

As shown in Table 1, the statements with line numbers 14 and 15 belong to a basic block, and the two lines of code are equally doubtful as calculated using Tarantula, Ochiai, and Tarantula *. Collecting and calculating every statement information in a basic block is a repetitive work, so the basic block is studied as the basic unit of defect location.

In this study, the object of study is mainly C language, C language contains a choice of structure switch and looping mechanism, while, etc., need to extend the definition of the basic block. Locating a basic block is a sequence of statements that, in a high-level language, are executed sequentially and without interruption. As shown in Table 2, the division method of various basic blocks of structure in C language is given.

As shown in Table 2, the execution of the switch (…) select structure statement in the absence of deterministic input data does not determine whether the case was executed or not. Therefore, the switch (…) is divided into one basic block, and each case is divided into one basic block. For the code with nested program statement types, the code in the basic block is further partitioned according to the partitioning rules. If the case 1 in

the switch statement contains an if structure statement, the code in location block 2 is divided again according to the partitioning rules, dividing the location block 2 into five basic blocks.

Table 2. C basic block partitioning table

Sequential structure	if	switch	for	while
A…; B…; C…;	If(…) 1	Switch() 1	for(…) 1	while() 1
	{… 2}	case1: {… 2}	{… 2}	{… 2}
-	else {… 3}	case2: {… 3}	–	–

In the unit level defect location, dividing the whole code into basic location blocks can effectively merge the repetition information in the coverage matrix, simplify the coverage matrix and reduce the repetition suspicious calculation process. The basic block partition of the find Second function results $\left\{ \begin{matrix} \{2\}, \{3\}, \{4, 5\}, \{7, 8\}, \{10\}, \{11, 12\}, \{13\}, \\ \{14, 15\}, \{17\}, \{19\}, \{20\}, \{21\}, \{22\}, \{24\} \end{matrix} \right\}$ in a simplification of the cover matrix of 24×17 to that of 24×14. In addition, the length of chromosome coding can be reduced and the search efficiency can be improved by using ELT to search defect combinations.

2.3 Genetic Algorithm for Searching Optimal Population

Optimizing reinforcement learning algorithm with genetic algorithm, so genetic algorithm is used to search the optimal population and obtain the combination of defects in software testing. This section is divided into four stages, namely, hitting set, chromosome coding, fitness value function and genetic operator, the specific content of each stage as shown below.

Stage One: Hitting Set.

For a set cluster $Z = \{S_1, S_2, \cdots S_n\}$, if the set S_H satisfies $S_1 \cap S_H \neq \emptyset$, $S_2 \cap S_H \neq \emptyset$,…$S_n \cap S_H \neq \emptyset$, then the set S_H is said to be the hitting set of the set cluster Z. If S_H any subset of a set is not the hitting set of a set cluster Z, it is the smallest hitting set S_H of the set cluster Z.

For a set $F = \{f_1, f_2, \cdots f_n\}$ of location-based blocks, if all failed test cases F have at least one location-based block f_i in execution, then F is a defect combination. For a program that contains defects, there are multiple combinations of defects $F_{accuracy}$. In all combinations of defects, there is only one combination of defects that contains all defects. This combination of defects is called. Is the accurate solution of defect location, defect location is approximation $F_{accuracy}$.

In a test case, the failed use case overrides the information set cluster $L = \{C_1, C_2, \cdots C_n\}$, where the locating base block C_i representing the i failed use case overrides the information set. Before locating defects, it is necessary to be able to explain all

the failure test cases, that is, each failure test case must cover at least one location-based block, so the defect combination is the hitting set of the failure use case coverage F information set. In this study, the failure case overrides the hitting set of information set clusters L as a guess of defect combination. The main step of solving the hitting set of the information set covered by the failed use cases is to form a hitting set by a set, and construct the hitting set for each element in the set; a new set is added to the hitting set, and if the previous hitting set is not the hitting set of the new set, a new element is added to the hitting set, and the new element is an element that belongs to the new set but does not belong to the previous set cluster; until all the failed test case collections are contained in the set cluster. In solving the hitting set, subtract in advance the hitting set consisting of the basic block defining the variable [6]. Because part of the code defined by a variable will be run every time the test case is run, it is not reasonable to guess the defects in the definition of the variable.

Stage Two: Chromosome Coding.

Genetic algorithms represent the distribution of candidate defects as a binary vector:

$$C = \{c_1, c_2, \cdots c_n\} \tag{1}$$

In formula (1), the length n of the binary vector represents the j number of executable entities in the program under test, and the value range of the first position is 0 or 1. At that time $c_j = 1$, it indicates that the j entity of the candidate distribution represented by the vector has defects; at that time $c_j = 0$, it indicates that the j entity of the candidate distribution represented by the vector does not have defects. For example, if $C = \{0, 0, 1, 0, 0, 0, 0, 1, 0\}$ are 9 executable entities in the candidate distribution, it assume that there are defects in the second and seventh entities and no defects in the other entities.

Stage 3: Fitness value function.

The Tarantula and Ochiai formulae for the calculation of the degree of suspicion factor derive the degree of suspicion factor from the calculation of the spectrum information of the test cases and the results of the execution, thereby deriving the degree of suspicion value for each program entity [7]. Then, the ranking list can be obtained by sorting the suspicious values of the program entities from high to low, and the developer can check the program in turn according to the list to find out the specific position of the defects in the software program.

The calculation formulas of Tarantula and Ochiai measure the degree of suspicion of each statement, and this kind of method has a good effect in the case of single defect. However, in the actual process of software debugging, the program under test often contains many defects. Tarantula and Ochiai's list of suspicious rankings can be used as a reference in the case of multiple defects, but because they are not optimized for multiple defects, there are potential problems. The Multi-Ochiai formula used in this study is an improvement on the Ochiai formula for calculating the Ochiai suspicious coefficient [8].

When there are multiple bugs in a program, there is a big difference from the assumption of a single bug program, so the method of this study is based on the following assumption.

Hypothesis 1: The skepticism of a defect distribution is proportional to the number of failed use cases that the distribution can interpret.

Hypothesis 2: The degree of doubt about a defect distribution is inversely proportional to the number of use cases through which the distribution can be interpreted.

Hypothesis 3: Hypothesis 1 accounts for a larger proportion of the set suspicion formula.

In the enhanced learning, using Multi-Ochiai formula to evaluate a candidate defect distribution hypothesis C. The more the candidate distribution can explain, the higher the suspect value is, and the more the candidate distribution is covered by the correct test case, the lower the suspect value is. At the same time, the number of defects included in the candidate distribution assumption is also a consideration index. Under the same coverage condition, the lower the number of defects, the higher the suspicious value. Formula (2) defines the calculation method of the Multi-Ochiai formula for calculating suspicious values:

$$MultiOchiai = \frac{\varphi(c)}{\sqrt{|T_P \times (\varphi(c) + P(c))|}} \tag{2}$$

In Formula (2), $\varphi(c)$ is the ability to interpret a failed test case for a candidate defect distribution is actually the summation of the coincidental numbers of the program entity coverage of the failed test case and the location of the existing defect in the candidate defect distribution; $P(c)$ is the ability of the defect distribution to interpret the correct test case.

Stage 4: Genetic operators.

The selection operator is used to select the individuals that need to be cross-mutated and to determine the number of offspring that will eventually be generated. There are many selection operators in common use. There are random selection operators and roulette selection operators. In general, the higher the fitness value of an individual is, the more likely it is to contain a better gene. Therefore, the roulette selection operator is used in augmented learning to select the operated individual [9]. Roulette selection operator is used to simulate the selection of individuals using the roulette wheel used in gambling, and the fitness value of each individual is converted to the probability of selection in proportion. The whole roulette wheel can be divided into N_p sectors, and the percentage of the area of each individual sector to the total area of the circle shall be calculated by formula (3). In the process of selection, need to set a parameter $GGAP$ to specify the proportion of the selected individuals in the population, so it need to select the $GGAP \times N_p$ number of times to get the individuals. To indicate the $GGAP \times N_p$ number of unselected individuals in the population at the time of the previous selection. N_{nc} To indicate an unselected individual, C_1 is the probability that an individual C will be selected in that selection may be calculated as follows:

$$P(c) = \frac{MultiOchiai(c)}{\sum_{i=1}^{N_{nc}} MultiOchiai(c_i)} \tag{3}$$

The crossover operator acts on the two selected individuals to generate new offspring by gene exchange between them. it chooses shuffle crossover operator to apply to the crossover process of genetic algorithm because of the irregularity of many defects in the program. In the process of crossover, parameters are set P_c to describe the probability

of gene exchange between two individuals at a certain location, and shuffle crossover operator allows crossover at all locations.

Because of the limitation of space, the mutation operator and the re-insertion step are not discussed again.

2.4 Identify Multiple Defect Locations

In the stage of determining multiple defect locations, the candidate distribution population obtained by genetic algorithm is transformed into the corresponding ranking of suspicious values of real program entities. First of all, the candidate defect distribution individuals are sorted according to the order from high to low, and the suspect value of the higher individual is, the more likely it is to contain defect location [10–13]. So the final sequencing of program entities will be selected from the candidate distribution in order from the first to the last. If the same candidate distribution contains more than one defect location, the location of these defects will be ranked randomly. For example, the following is a sorted candidate defect distribution population:

$$\left\{\begin{array}{l}\langle 0,0,0,1,0,1,0,0,0,0\rangle \\ \langle 0,1,0,1,0,0,0,0,0,0\rangle \\ \langle 0,1,0,0,0,1,0,0,0,0\rangle \\ \langle 1,0,1,0,0,0,1,1,0,1\rangle \\ \langle 1,0,0,0,1,0,0,1,1,0\rangle \end{array}\right\} \tag{4}$$

From top to bottom, the suspicious value of each candidate distribution decreases in turn, and the defect location in the hypothesis of the candidate distribution nearer to the candidate distribution is more likely to be practical. Therefore, the fourth and sixth program entities of the first candidate distribution have the highest degree of suspicion, and these two program entities are ranked in a random manner, followed by a sequence of program entities selected from the subsequent candidate distribution. The list of the degree of suspicion that can be obtained from this candidate defect distribution is $\langle e4, e6, e2, e1, e8, e3, e7, e10, e5, e9\rangle$. Because a random strategy is used in the selection of defect distributions with multiple defect assumptions, the ranking of statements by the same candidate distribution population may not be unique.

Through the above process, the simultaneous positioning of multiple defects in software testing is realized, which provides better method support for software application and development.

3 Experiment and Result Analysis

3.1 Evaluation Dataset Preparation

The experimental study used both small-scale and large-scale programs that can be downloaded from the SIR library. Of these, (1) small-scale programs come from seven of the Siemens suites, which have a minimum of 174 lines and a maximum of 539 lines, more than half of which are executable statements. Each program has one correct version

and multiple incorrect versions, each of which contains only one defect. These programs are packaged with a minimum of 1052 test cases and a maximum of 5,542 test cases. (2) Large-scale programs from Linux programs are gzip, grep, and sed. These three programs are 6576 lines, 12635 lines and 7125 lines, of which executable statements accounted for about 1/4. But the set of test cases that these programs come with is small, with a minimum of 213 and a maximum of 470.

Some of the bugs in the wrong version of the Siemens program are not executable, which is beyond the scope of the SFL approach and makes it difficult to get accurate results. Therefore, some defects in the experiment need to be reimplanted. It e implanted a single defect in seven of Siemens' programs to perform a single defect evaluation experiment, and in four of them implanted multiple defects to perform multiple defect evaluation experiments. Selecting four programs to implant multiple defects requires that it contain more lines of executable code to implant more combinations of defects. The three Linux source code implants were all in executable lines of code, so combined

Table 3. Schedule of evaluation procedures

Program under test	Multi defect version	Number of test cases
Print_tokens	32	4130
Print_tokens2	34	4115
Replace	45	5542
Schedule	13	2650
Schedule2	15	2710
Tcas	18	1608
Tot_info	48	1052
Gzip	29	213
Grep	15	470
sed	3	360
Program under test	Multi defect version	Number of test cases
Print_tokens	32	4130
Print_tokens2	34	4115
Replace	45	5542
Schedule	13	2650
Schedule2	15	2710
Tcas	18	1608
Tot_info	48	1052
Gzip	29	213
Grep	15	470
sed	3	360

the bugs to produce a batch of two bugs and three bugs, based on a single bug version. Specific information about the evaluation process in the empirical study is shown in Table 3.

3.2 Selection of Evaluation Indicators

In single defect location, an index *EXAM* is usually used to evaluate the validity of the method. The index returns the percentage of all statements that need to be checked before the defect statement is detected. For a given program under test, the smaller the value, the better the effect of defect location is. But in the multi-defect localization, the single defect target is not completely suitable. Therefore, extend the definition to the evaluation of multi-defect localization methods, which mainly includes $EXAM_F$ and $EXAM_L$, which $EXAM_F$ returns the percentage of all statements that need to be checked before the first defect is detected, and $EXAM_L$ returns the percentage of statements that need to be checked before the last defect is detected.

3.3 Analysis of Experimental Results

The evaluation indexes are shown in Table 4.

Table 4. Table of indicators of evaluation

Number of experiments	$EXAM_F$		$EXAM_L$	
	Proposed method	Existing methods	Proposed method	Existing methods
1	1.23	2.56	0.95	1.85
2	0.98	2.00	0.85	1.87
3	0.78	2.01	0.88	1.90
4	0.80	1.98	0.78	2.04
Average value	0.9475	2.1375	0.8650	1.9150

As shown in Table 4, compared with the average evaluation index values of existing methods, the proposed method $EXAM_F$ reduces 1.19 and $EXAM_L$ reduces 1.05, which fully indicates that the proposed method has better positioning performance.

4 Concluding Remarks

Based on Reinforcement Learning, this paper presents a new method of simultaneous defect location in software testing, which greatly reduces the evaluation index $EXAM_F$ and $EXAM_L$, and provides more effective guarantee for software application.

References

1. Qian, H., Tong, H., He, M.Z., et al.: Observation of carrier localization in cubic crystalline Ge2Sb2Te5 by field effect measurement. Sci. Rep. **8**(1), 486 (2018)
2. Hao, Z., Bechtel, H.A., Kneafsey, T., et al.: Cross-scale molecular analysis of chemical heterogeneity in shale rocks. Sci. Rep. **8**(1), 2552 (2018)
3. Lee, S., Lee, J., Ryu, B., et al.: A micromechanics-based analytical solution for the effective thermal conductivity of composites with orthotropic matrices and interfacial thermal resistance. Sci. Rep. **8**(1), 7266 (2018)
4. Ohara, S., Gonçalves dos, J., Angelotti, J.A.F., et al.: A multisystem for multicomponent on a blend of industrial agroeous wastes for the simultaneous manufacturing of industrials for solid-by state. Food. Technol. **38**(1), 131–137 (2018)
5. Naharros, I.O., Cristian, F.B., Zang, J., et al.: The ciliopathy protein TALPID3/KIAA0586 acts upstream of Rab8 activation in zebrafish photoreceptor outer segment formation and maintenance. Rep. **8**(1), 2211 (2018)
6. van den Heuvel, Corina, N.A.M., Das, A.I., De Bitter, T., et al.: Quantification and localization of oncogenic receptor tyrosine kinase variant transcripts using molecular inversion probes. Rep, **8**(1), 7072–7072 (2018)
7. Fu, W., Liu, S., Srivastava, G.: Optimization of big data scheduling in social networks. Entropy **21**(9), 902 (2019)
8. Liu, S., Li, Z., Zhang, Y., et al.: Introduction of key problems in long-distance learning and training. Mob. Networks Appl. **24**(1), 1–4 (2019)
9. Liu, S., Liu, D., Srivastava, G., et al.: Overview and methods of correlation filter algorithms in object tracking. Complex Intell. Syst. (2020). https://doi.org/10.1007/s40747-020-00161-4
10. Kuo, D.H., Abdullah, H., Gultom, N.S., et al.: Ag-decorated mosx laminar-film electrocatalyst made with simple and scalable magnetron sputtering technique for hydrogen evolution: a defect model to explain the enhanced electron transport. ACS Appl. Mater. Interfaces **12**(31), 35011–35021 (2020)
11. Mao, X., Chow, J.K., Tan, P.S., et al.: Domain randomization-enhanced deep learning models for bird detection. Sci. Rep. **11**(1), 639 (2021)
12. Beloborodov, D., Ulanov, A.E., Foerster, J.N., et al.: Reinforcement learning enhanced quantum-inspired algorithm for combinatorial optimization. Mach. Learn. Sci. Technol. **2**(2), 025009 (12pp) (2021)
13. Cai, M., Jiang, Y., Gao, C., Li, H., Yuan, W.: Learning features from enhanced function call graphs for Android malware detection. Neurocomputing **423**(2), 301–307 (2021)