Research Article

Automatic Generation of Test Cases Based on Genetic Algorithm and RBF Neural Network

Zhenpeng Liu 1, Xianwei Yang 1, Shichen Zhang 1, Yi Liu 1, Yonggang Zhao 2, and Weihua Zheng 3,4

1 School of Cyberspace Security and Computer, Hebei University, Baoding, China
2 School of Management Engineering and Business, Hebei University of Engineering, Handan, China
3 School of Information Engineering, Handan University, Handan 056005, China
4 School of Information Science and Engineering, Xinjiang University of Science and Technology, Kuerle 841000, China

Correspondence should be addressed to Weihua Zheng; zhwh369@hdc.edu.cn

Received 16 November 2021; Revised 20 April 2022; Accepted 2 May 2022; Published 23 May 2022

Academic Editor: Demian D’Mello

Copyright © 2022 Zhenpeng Liu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Software testing plays an important role in improving the quality of software, but the design of test cases requires a lot of manpower, material resources, and time, and designers tend to be subjective when designing test cases. To solve this problem and make the test cases have objectivity and greater coverage, a branch coverage test case automatic generation method based on genetic algorithm and RBF neural network algorithm (GAR) is proposed. In terms of test case generation, based on the genetic algorithm optimized in this paper, a certain number of test case samples are randomly selected to train the RBF neural network to simulate the fitness function and to calculate the individual fitness value. The experiment uses 7 C language codes to automatically generate test cases and compares the experimental data generated by the branch coverage test case generation method based on adaptive genetic algorithm (PDGA), traditional genetic algorithm (SGA), and random test generation method (random) to evaluate the proposed algorithm. The experimental results show that the method is feasible and effective, the branch coverage is increased in the generation of test cases, and the number of iterations of the population is less.

1. Introduction

Software testing is an important means to ensure the quality and reliability of the software [1]. The testing phase is the most important phase of the software. It can reflect the performance of the software, but it occupies nearly half of the development cost [2, 3]. The design of test cases requires a lot of manpower, material resources, and time [4]. Therefore, scholars use automated methods to generate test cases to reduce the huge cost of software testing.

In 1995, Jones et al. [5] applied a genetic algorithm to the automatic generation of test data for the first time, and the genetic algorithm is superior to the traditional random algorithm in terms of the number of iterations. Lin and Yeh [6] used genetic algorithms to automatically generate test cases based on path coverage. Singh et al. [7] proposed an automatic mutation testing tool for aspect-oriented programs, which implements different mutation operations to deal with faults. Literature [8, 9] uses a genetic algorithm to generate test data for data flow coverage criteria. Raajapaa et al. [10] used a combination of genetic algorithm and graph theory to generate test cases. In the fitness function value of the traditional genetic algorithm, only the information of the predicate conditions is considered, so the search space lacks effective guidance information, the positioning of the global minimum lacks effective information, and as a result, the search speed is slow. It may converge to the local optimal solution and there is a premature problem. Singh et al. [11] proposed a multiobjective ant lion optimization algorithm based on the multiobjective optimization problem covering test data and discussed how the algorithm can improve path coverage by reducing the number of tests. Mateen et al. [12] proposed the use of genetic optimization test case generation algorithm, which generates test cases through mutations in...
the program and uses mutation scores to optimize the test cases. When the mutation score is less than a certain percentage, it used the remaining test cases to perform genetic algorithm operations. Gupta [13] proposes a two-phase algorithm considering test case selection and test case prioritization techniques for performing regression testing on multiple modules ranging from small lines of code to huge working languages.

Yao et al. [14] proposed a genetic algorithm and BP neural network path coverage test case generation method, which makes good use of the BP neural network model to calculate the fitness value of an individual. Ji et al. [15] proposed a test case generation based on BP neural network for data flow testing and used a genetic algorithm to improve the test cases. Based on the genetic algorithm, Chahar et al. [16] described, analyzed, and applied various aspects of the development of genetic algorithms in detail. Srinivas and Patnaik [17] proposed adaptive probabilities of crossover and mutation in genetic algorithms (AGA), which uses adaptive crossover and mutation probabilities to achieve the dual goals of maintaining population diversity and genetic algorithm convergence in adaptive genetic algorithms. Based on the adaptive genetic algorithm proposed by Srinivas et al., Yang et al. [18] proposed an improved adaptive genetic algorithm for function optimization (IAGA), which improves the crossover probability and mutation probability of the traditional adaptive genetic algorithm and solves the problem of slow convergence and poor stability based on traditional genetic algorithms and other issues. However, the improved algorithm has poor performance in terms of convergence speed and time to find the optimal solution. Song et al. [19] proposed an improved real-coded genetic algorithm, which can effectively avoid the problems of premature convergence and falling into a local optimal state when solving complex numerical function optimization problems. However, the algorithm is not ideal in terms of test case generation.

In response to the above problems, this paper proposes a branch coverage test case automatic generation method based on the genetic algorithm and the RBF neural network algorithm (GAR). The main contributions of this paper are as follows:

1. The RBF neural network is used to simulate the fitness function of the branch coverage path to solve the fitness value of the population.
2. The mutation probability in the mutation operation of the genetic algorithm is optimized, and the problems of premature, slow convergence, and poor stability of the previous genetic algorithm are solved.
3. The crossover probability in the crossover operation of the genetic algorithm is optimized to solve the problems of premature, slow convergence speed, and poor stability of the adaptive genetic algorithm.
4. This paper uses 7 classic C language codes and compares them with the experimental data of branch coverage test case generation method based on adaptive genetic algorithm (PDGA), traditional genetic algorithm (SGA), and random test generation method (Random). The experimental data of the algorithm is compared to verify the feasibility and effectiveness of the proposed method.

2. The Construction of Genetic Algorithm Parameters Generated by Test Cases

Genetic algorithm is a global optimization algorithm. Compared with other group of intelligent optimization algorithms, genetic algorithm has a good global search ability [20]. In the optimization process of traditional genetic algorithms, the relevant parameters of the algorithm are fixed. Under the background that the group is constantly adjusted to external factors, the fixed parameters cannot meet the dynamic needs of individuals in different processes, which affects the performance and efficiency of the algorithm [21].

2.1. Construct Fitness Function Based on RBF Neural Network

When the traditional genetic algorithm automatically generates test cases, the calculation of fitness value requires the program instrumentation. However, the program instrumentation requires a lot of work, and the calculation of the fitness value of each individual is based on the instrumentation program, which seriously affects the speed of test case generation. Constructing the fitness function to calculate the individual fitness value of the test case can well improve the negative efficiency brought by the instrumentation program. The construction of fitness function will directly affect the direction of population development. Thus, the construction of fitness function is extremely important.

In the generation of genetic algorithm test cases, the construction method of fitness function often uses layer proximity [14] and branch distance [22]. The level of proximity indicates the degree of deviation between the path taken by the test case and the target path; the branch distance indicates the degree of satisfaction with the condition that a predicate is true or false.

The fitness function is constructed as formula (1), which generates test cases and transforms them into solving the problem of the minimum value of the objective function.

$$f(x) = \text{approach}_\text{level}(x) + 1 - 1.001^{-\text{branch}_\text{dist}(x)}, \quad (1)$$

where $\text{approach}_\text{level}(x)$ is the layer proximity and $1 - 1.001^{-\text{branch}_\text{dist}(x)}$ is the branch distance.

Assuming that under the condition “if (a > b),” the value of branch distance function branch_dist(x) is as shown in the following formula:

$$\text{branch}_\text{dist}(x) = \begin{cases} b - a + \varepsilon, & b \geq a, \\ 0, & a > b. \end{cases} \quad (2)$$

In this conditional statement, when the branch distance is true, the value of the conditional branch distance is 0, and when it is false, the value of the conditional branch distance is $b - a + \varepsilon$, where $\varepsilon$ is an appropriate penalty.
Assuming that under the condition “if \((a = b)\),” the value of branch distance function \(\text{branch\_dist}(x)\) is as shown in the following formula:

\[
\text{branch\_dist}(x) = \begin{cases} 
|b - a|, & a \neq b, \\
0, & a = b.
\end{cases}
\] (3)

Assuming that under the condition “if \((a \neq b)\),” the value of branch distance function \(\text{branch\_dist}(x)\) is as shown in the following formula:

\[
\text{branch\_dist}(x) = \begin{cases} 
0, & a \neq b, \\
1, & a = b.
\end{cases}
\] (4)

Assuming that under the condition “if \((a)\),” the value of branch distance function \(\text{branch\_dist}(x)\) is as shown in the following formula:

\[
\text{branch\_dist}(x) = \begin{cases} 
0, & a = \text{True}, \\
1, & a = \text{false}.
\end{cases}
\] (5)

The GAR algorithm uses the RBF neural network to simulate the calculation process of formula (1), uses it as a fitness function to obtain the fitness value of the individual test case, and solves the test case through the instrumentation program to solve the fitness value. The RBF neural network performs unsupervised learning in the first stage. It determines the center vector and variance of each node in the hidden layer according to the input samples. In the second stage, it performs supervised learning and determines the weights from the hidden layer to the output layer according to the size of the sample value. The RBF neural network structure is shown in Figure 1. The main construction work of the neural network structure is to determine the center of the radial basis function, the variance, and the weights from the hidden layer to the output layer. In the RBF neural network, the center of the radial basis function, the variance, and the weights from the hidden layer to the output layer are determined as follows.

The determination of the center of the radial basis function: in the RBF neural network, a radial basis function is established for each sample of input data and output data \(\phi(\|x, x^m\|)\), where \(x\) represents a vector and \(x^m\) represents the center of the radial basis function. Steps to determine the center of the radial basis function: first, the centers are initialized, that is, \(m\) training samples are randomly selected as the center point. Subsequently, \(n\) training samples are randomly selected, and the distance between the samples and the center is calculated. Next, the input sample closest to the center point is searched. Finally, the cluster centers are adjusted and determined whether the center has changed; otherwise, it is urgent to iterate.

The determination of variance: when the center is determined, the variance of the radial basis function is determined according to the center point. The variance of the radial basis function can be expressed as \(\sigma^2 = (\Delta_{\text{max}} / \sqrt{2n})^2\), where \(\Delta_{\text{max}}\) represents the maximum distance between the centers.

The determination of the weights from the hidden layer to the output layer: the least mean square algorithm is used. When the square difference between the actual output of the neuron and the expected output is the smallest, it is determined to be the target value, and the objective function is \(a\), where \(b\) is the expected value. The adjusted weight vector is \(\Delta W_i = \eta (b_i - W_i^T x) x\), where \(\eta\) the learning step size and the value range are \(0 \leq \eta \leq 1\).

Evaluation of the individual fitness function of the construction test case of the RBF neural network:

1. Suppose there are \(L\) test cases \((X = (x_1, x_2, \ldots, x_i)^T)\), which are input vectors
2. The samples in the input layer are calculated to the hidden layer of the neural network, and the output in the hidden layer is \(C = (c_1, c_2, \ldots, c_n)^T\)
3. The weights \(\Delta W_i\) are added to the hidden layer, and the individual fitness value of the test case is outputted

2.2. Optimization of Crossover Probability and Mutation Probability. The crossover operation is to use the genes in the parent to generate the next generation and generate new individuals [23]. The crossover operation in the genetic algorithm is mainly to select the better genes in the parent generation for hybridization, which can ensure that the new individuals generated have better genes than the previous generation, and is beneficial to improve the search efficiency of the genetic algorithm.

In the mutation operation, not only the mutation operator but the mutation probability also plays a very important role. The mutation operation in the genetic algorithm is the main method to improve the diversity of the population [24]. The mutation operation performs a specific replacement of the code on the individual’s gene to achieve the purpose of forming a new individual. The main purpose of the mutation operator in the genetic algorithm is as follows: on the one hand, when the algorithm gets the neighborhood close to the optimal solution, the process of obtaining the optimal solution is speeded up; on the other hand, it can prevent the algorithm from premature convergence, generate new individuals, and maintain
When genetic algorithms lose diversity due to selection operations, the mutation process can act as a remedy for genetic diversity. When the mutation probability is large, the genetic algorithm will become a traditional random search algorithm, which will cause the population to evolve slowly and lose its original meaning. When the mutation probability is small, the purpose of efficient search will not be achieved.

In the traditional genetic algorithm, in the process of genetic operation, the crossover operation is generally performed first, and then, the mutation operation is performed. This genetic manipulation method will have a very ideal effect in the early stage, but in the later stage, the fitness value of the entire population will be very close. When the fitness values of the individuals are relatively close, the crossover operation produces new individuals with large changes, and the subsequent mutation operation is easy to destroy the optimal individual, thereby slowing down the convergence speed. The improved genetic algorithm judges the sequence of crossover and mutation operations according to the distribution of individual fitness values in the population to enhance the diversity of individuals in the population and pave the way for the next generation of genetic operations.

Based on the insufficient crossover probability and mutation probability in the traditional genetic algorithm, the steps of genetic operation in the genetic algorithm used in this paper are as follows:

1. Passing in the fitness value and judging the distribution of individual fitness values through the function \( \arcsin \left( \frac{f_{\text{avg}}}{f_{\text{max}}} \right) \) as the \( f_{\text{avg}} \) increases, \( \arcsin \left( \frac{f_{\text{avg}}}{f_{\text{max}}} \right) \) grows faster than the linear function, so as to better judge the concentration of the population. When the absolute value of the function is greater than or equal to \((\pi/6)\), it can be judged that the fitness of the newly generated test case population is relatively concentrated; when the absolute value of the function is less than \((\pi/6)\), it can be judged that the fitness value of the test case population is relatively scattered.

2. When the function \( |\arcsin \left( \frac{f_{\text{avg}}}{f_{\text{max}}} \right)| \geq (\pi/6) \), firstly it performs the selection operation, then the mutation operation, and finally the crossover operation.

3. When the function \( |\arcsin \left( \frac{f_{\text{avg}}}{f_{\text{max}}} \right)| < (\pi/6) \), firstly it performs the selection operation, then the crossover operation, and finally the mutation operation. The steps of genetic operation in a genetic algorithm are shown in Figure 2.

In terms of crossover probability, the importance of the individual fitness value in the mutation probability is reflected, and the selected individual fitness value will not make a big difference within the upper and lower range of the average fitness value when the maximum fitness among the two selected individuals is less than the average fitness value of the population, the maximum fitness value in the population is used to subtract the average value of the two individual fitness values selected, and the mutation probability is as shown in the following formula:

\[
p_c = \begin{cases} 
  k_1 \frac{f_{\text{max}} - f}{f_{\text{max}} - f_{\text{avg}}}, & f \geq f_{\text{avg}} \\
  \frac{f_{\text{max}} - f_c}{f_{\text{max}} - f_{\text{avg}}}, & f < f_{\text{avg}} 
\end{cases}
\]  

(6)

where the value range of \( k_1 \) is \((0, 1]\), \( f_{\text{max}} \) is the maximum individual fitness value in the population, \( f \) is the maximum fitness value of the two individuals to be crossed, \( f_{\text{avg}} \) is the average fitness of the population, and \( f_c \) is the average fitness value of the two individuals to be crossed.

In terms of mutation probability, a power function is used to solve the individual’s mutation probability. It can well reflect the importance of the individual fitness value in the crossover probability, and the selected individual fitness value will not make a big difference within the upper and lower range of the average fitness value. The mutation probability is as shown in the following formula:

\[
p_m = k_2 \left( e^{(1/10)} - e^{(f_i/10)} \right),
\]  

(7)
where $f_i$ is the fitness value of the selected individual to be mutated, and the value range of $k_2$ is $(0, 5)$.

3. Branch Coverage Test Case Generation Algorithm

The steps of the branch coverage test case automatic generation method based on genetic algorithm and RBF neural network algorithm are as follows:

1. In the first stage, after the neural network training module, a certain number of test case samples are selected and initialized.
2. The center and variance of the radial basis function neural network are determined.
3. The weights of the RBF neural network are determined, and the weights are corrected.
4. The individual fitness value is calculated and outputted.
5. In the neural network module, the error of the individual fitness value is judged whether the condition is met; if it is not met, go to step 3, and if it is met, it will enter the genetic algorithm module.
6. In the second stage, in the genetic algorithm module, the test case population is randomly generated, the branch coverage is recorded according to the condition of the branch coverage, the branch coverage is reduced, a list is created to store the uncovered branches, and the test case population is initialized.
7. Using the trained neural network module, the fitness value of each test case is calculated.
8. The condition is judged whether it is met (according to the required judgment condition set by the program), and when the condition is met, the test case covered by the branch is output.
9. When the conditions of 8 are not met, the individual fitness value distribution of the test case is judged.
10. Assuming that the conditions of step 9 are met, it needs to be performed in the order of selection operation, mutation operation, and crossover operation. Then, go to step 7.

Figure 3: Automatic generation of test cases based on genetic algorithm and RBF neural network.
(11) Assuming that the conditions of step 9 are not met, it needs to be performed in the order of selection operation, crossover operation, and mutation operation. Then, go to step 7.

The flow chart of test case generation based on genetic algorithm and RBF neural network is shown in Figure 3.

4. Algorithm Analysis and Verification

For the experimental genetic algorithm parameters and genetic operation settings in the experiment, the value of $k_1$ is 0.4, and the value of $k_2$ is 1. The chromosomes use binary coding, the selection operation is roulette selection, the crossover operation is a single point crossover, and the mutation is a single point mutation.

4.1. Test Case Generation and Verification of Branch Coverage.

In terms of branch coverage test case generation, the experiment used 7 C programs [25–27] to verify the feasibility and effectiveness of the test case generation method based on the genetic algorithm and RBF neural network proposed in this paper. 7 C programs used are Gcd, triangle, remainingSth, and numbers program structure in the appendix. Among them, the Bessel function (beseel), the calculation of the day of the week (calday), and the arithmetic function of the calculation of complex numbers (complex) are from the book quoted from Numerical recipes in C: The art of scientific computing [28] by W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, and others edited. The details of the tested programs are shown in Table 1.

In order to better visualize the advantages of the test case generation algorithm (GAR) proposed in this article, we will compare the following three methods for branch coverage test case generation—the branch coverage test case generation method based on the adaptive genetic algorithm (PDGA), traditional genetic algorithm (SGA), and random test generation method (Random). The evaluation criteria of the experiment are average branch coverage rate, maximum branch coverage rate, and average convergence algebra.

The branch coverage rate is calculated as in the following formula:

$$ t = \frac{n}{m} \times 100\%, $$

where $t$ represents the branch coverage rate, $n$ represents the number of evaluation results, and $m$ represents the total number of judgment results.

In the experiment, the common parameters of the four branch coverage test case generation methods are set as follows: the population size is 50, and the maximum number of evolutionary iterations is 100. To avoid randomness from adversely affecting the progress of the experiment, this paper will run 50 times in each method for each program under test. The evaluation criteria of the experiment are average branch coverage rate, maximum branch coverage rate, and average convergence algebra.

The average branch coverage rate ($Ac$) is calculated as in the following formula:

$$ Ac = \frac{\sum t_{(1,2,...,n)}}{n}, $$

where $Ac$ represents the average branch coverage rate and $t$ represents the branch coverage rate of the program under test after it is executed in the algorithm.

Maximum branch coverage rate (MaxC): the best branch coverage rate of the program under test after running multiple times in an algorithm.

Average convergence algebra (Ag): when the tested program completes a complete branch coverage or each time it converges to the optimal solution in the algorithm, the number of iterations is added, and the final total number of iterations is divided by the number of executions of the program under test to get the average convergence algebra (Ag).

The experimental results of the average coverage rate and the maximum branch coverage rate of the 7 tested programs after running 50 times in the four algorithms are shown in Table 2.

It can be seen from Table 2 that the maximum branch coverage rate of the four algorithms for the program p1 (gcd) has reached 100% coverage, but the average branch coverage rate of the Random algorithm has not reached 80%. In the program p2 (triangle), the maximum branch coverage rate of the three genetic algorithm-based test case automatic generation methods all reached 100%, while the Random algorithm did not reach 80%. In the average branch coverage rate, the SGA algorithm reached 93.80%, but the Random algorithm only reached 55.60%. In the programs p3, p4, and

| Program name | Number of input parameters | Number of branches | Number of lines of code | Program information                      |
|---------------|----------------------------|------------------|-------------------------|------------------------------------------|
| gcd           | 2                          | 8                | 28                      | Greatest common denominator              |
| triangle      | 3                          | 28               | 57                      | Classic triangle classification program   |
| beseel        | 2                          | 21               | 245                     | Bessel function                          |
| calday        | 3                          | 22               | 72                      | Calculate the day of the week             |
| remainderSth  | 2                          | 18               | 49                      | Calculate division of quotient and reminder |
| numbers       | 1                          | 28               | 71                      | Parse large numbers from integers to strings |
| complex       | 6                          | 24               | 74                      | Calculate complex arithmetic functions    |

Table 1: Experimental objects.
p5, the maximum branch coverage rate of the PDGA, SGA, and GAR algorithms reached 100%, while Random was 76.19%, 91.91%, and 83.33%. In program p7, there are only two algorithms, PDGA and GAR, whose average branch coverage and maximum branch coverage reached 100%, while SGA and Random both reached 100%. It can be found that the average branch coverage rate and the maximum branch coverage rate of the SGA algorithm are greater than those of the Random algorithm. In program p6, the method proposed in this paper (GAR) achieves 100% maximum coverage, 99.70% average branch coverage, and 98.23%
average branch coverage for the PDGA algorithm. It can be seen that when the number of branches reaches a certain level, the PDGA algorithm appears instability, but the method proposed in this paper is relatively stable.

From Figures 4 and 5, it can be found that the automatic generation method of test cases based on the genetic algorithm is better than the random algorithm, and the performance of the genetic algorithm is still very impressive. The GAR algorithm proposed in this paper is better than the other three methods. Based on the above six, the program achieves 100% average coverage and maximum coverage, and only the average coverage of p6 reaches 99.7%, which proves the feasibility and effectiveness of the method proposed in this paper. To more clearly show the advantages of the algorithm proposed in this paper, the average convergence algebras of PDGA, SGA, and GAR are given in Table 3, and a more intuitive discount graph is given in Figure 6.

To more clearly show the advantages of the algorithm proposed in this paper, the average convergence algebra of PDGA, SGA, and GAR is given in Table 3, and a more intuitive discount graph is given in Figure 6.

It can be seen intuitively from Table 3 that the number of iterations of the PDGA and GAR algorithms is less than that of the SGA algorithm. In programs p4 and p7, it can be found intuitively that the number of iterations of the method proposed in this paper is less than that of the PDGA algorithm, and other programs are also slightly less than the PDGA algorithm, which shows the effectiveness and feasibility of the method proposed in this paper.

5. Conclusion

In terms of automatically generating program branch coverage test cases, this paper proposes a test case generation method based on the genetic algorithm and the RBF neural network. Based on the genetic algorithm, the RBF neural network is used to simulate the fitness function to calculate the fitness value of the population individuals and to optimize the crossover probability and mutation probability of the genetic algorithm. In experiments, we compare with PDGA, SGA, and Random algorithms, and the results show that the algorithm has good properties in branch coverage test case generation.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors’ Contributions

Zhenpeng Liu proposed an automatic generation test case model based on genetic algorithm and RBF neural network and made an overall revision to the paper in the later stage. Xianwei Yang gave detailed working steps based on the model proposed by Zhenpeng Liu and carried out experimental verification and written papers. Shichen Zhang proposed some corresponding techniques in experiments. Yi Liu and Yonggang Zhao provided the tested programs related to the experiments. Weihua Zheng guided the writing process of the paper, made important contributions in the process of supplementary experiments and revisions, and made important comments on the revision of the paper. All the authors have read and agreed to the published version of the manuscript.

Acknowledgments

This research was supported by the Natural Science Foundation of Hebei Province, China, under Grant no. F2019201427, and Fund for Integration of Cloud Computing and Big Data, Innovation of Science and Education of China, under Grant no. 2017A20004.

References

[1] A. Johanson and W. Hasselbring, “Software engineering for computational science: past, present, future,” *Computing in Science & Engineering*, vol. 20, no. 2, pp. 90–109, 2018.
[2] M. Khari, A. Sinha, E. Verdú, and R. G. Crespo, “Performance analysis of six meta-heuristic algorithms over automated test suite generation for path coverage-based optimization,” *Soft Computing*, vol. 24, no. 12, pp. 9143–9160, 2020.
[3] D. A. Hasan, B. K. Hussan, S. R. M. Zeebaree, D. M. Ahmed, O. S. Kareem, and M. A. M. Sadeeq, “The Impact of Test Case Generation Methods on the Software Performance: A Review,” *International Journal of Science and Business*, vol. 5, pp. 33–44, 2021.
[4] Y. Chen, Y. Gao, Y. Zhou, M. Chen, and X. Ma, “Design of an automated test tool based on interface protocol,” in *Proceedings of the 2019 IEEE 19th International Conference on Software Quality*, pp. 57–61, Reliability and Security Companion (QRS-C), Sofia, Bulgaria, July 2019.
[5] B. F. Jones, H.-H. Shhame, and D. E. Eyres, “Automatic structural testing using genetic algorithms,” *Software Engineering Journal*, vol. 11, no. 5, pp. 299–306, 1996.
[6] J. C. Lin and P. L. Yeh, “Automatic test data generation for path testing using Gas,” *Information Sciences*, vol. 131, no. 1–4, pp. 47–64, 2001.
[7] M. Singh, P. K. Gupta, and S. Mishra, “Automated test data generation for mutation testing using aspectJ programs,” in *Proceedings of the 2011 International Conference on Image Information Processing*, pp. 1–5, IEEE, Shimla, India, 2011.
[8] S. Varshney and M. Mehrotra, “Search-based test data generator for data-flow dependencies using dominance concepts, branch distance and elitism,” *Arabian Journal for Science and Engineering*, vol. 41, no. 3, pp. 853–861, 2016.
[9] P. R. Srivastava and T. Kim, “Application of genetic algorithm in software testing,” *International Journal of Recent Technology and Engineering*, vol. 3, no. 5, pp. 87–96, 2009.
[10] V. Rajappa, A. Biradar, and S. Panda, “Efficient software test case generation using genetic algorithm based graph theory,” in *Proceedings of the 2008 First International Conference on Emerging Trends in Engineering & Technology IEEE*, pp. 298–303, Nagpur, India, July 2008.
[11] M. Singh, V. M. Srivastava, and K. Gaurav, “Automatic test data generation based on multi-objective ant lion optimization algorithm,” in *Proceedings of the 2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics,*
[12] A. Mateen, M. Nazir, and S. Afzar, “Optimization of test case generation using genetic algorithm (ga),” *International Journal of Computer Application*, vol. 151, no. 7, pp. 6–14, 2016.

[13] P. K. Gupta, “K-Stp crossover method based on genetic algorithm for test suite prioritization in regression testing,” *Journal of Universal Computer Science*, vol. 27, no. 2, pp. 170–189, 2020.

[14] X. J. Yao, D. W. Gong, and B. Li, “Evolutional test data generation for path coverage by integrating neural network,” *Journal of Software*, vol. 27, no. 4, pp. 828–838, 2016.

[15] S. Ji, Q. Chen, and P. Zhang, “Neural network based test case generation for data-flow oriented testing,” in *Proceedings of the 2019 IEEE International Conference on Artificial Intelligence Testing (AITest)*, pp. 35-36, IEEE, Newark, CA, USA, April 2019.

[16] V. Chahar, S. Katoch, and S. S. Chauhan, “A Review on genetic algorithm: past, present, and future,” *Multimedia Tools and Applications*, vol. 80, pp. 8091–8126, 2021.

[17] M. Srinivas and L. M. Patnaik, “Adaptive probabilities of crossover and mutation in genetic algorithms,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 4, pp. 656–667, April 1994.

[18] C. Yang, Q. Qian, W. Feng, and M. Sun, “An improved adaptive genetic algorithm for function optimization,” in *Proceedings of the 2016 IEEE International Conference on Information and Automation (ICIA)*, pp. 675–680, Ningbo, China, August 2016.

[19] Y. Song, F. Wang, and X. Chen, “An improved genetic algorithm for numerical function optimization,” *Applied Intelligence*, vol. 49, no. 5, pp. 1880–1902, 2019.

[20] D. Liu, “Mathematical modeling analysis of genetic algorithms under schema theorem,” *Journal of Computational Methods in Science and Engineering*, vol. 19, no. S1, pp. 131–137, 2019.

[21] Y. Fang, X. Xiao, and J. Ge, “Cloud computing task scheduling algorithm based on improved genetic algorithm,” in *Proceedings of the 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 852–856, IEEE, Chengdu, China, March 2019.

[22] M. R. Rajkumari and B. G. Geetha, “Automatic test data generation using genetic algorithm and program dependence graph,” *Journal of Computer Applications*, vol. 3, no. 4, pp. 1–6, 2010.

[23] Z. H. Ahmed, “Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator,” *International Journal of Biometric and Bioinformatics*, vol. 3, no. 6, pp. 96–105, 2010.

[24] A. K. Das and D. K. Pratihar, “A direction-based exponential mutation operator for real-coded genetic algorithm,” in *Proceedings of the 2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)*, pp. 1–4, IEEE, Kolkata, India, 2018.

[25] E. Alba and F. Chicano, “Observations in using parallel and sequential evolutionary algorithms for automatic software testing,” *Computers & Operations Research*, vol. 35, no. 10, pp. 3161–3183, 2008.

[26] A. Bouchachia, “An immune genetic algorithm for software test data generation,” in *Proceedings of the 7th International Conference On Hybrid Intelligent Systems (HIS 2007)*, pp. 84–89, IEEE Computer Society, Kaiserslautern, Germany, 2007.