Research on Software Test Data Optimization Using Adaptive Differential Evolution Algorithm

Zheheng Liang[1,2(✉)], Wuqiang Shen[1,2], and Chaosheng Yao[1,2]

[1] Joint Laboratory on Cyberspace Security of China Southern Power Grid, Guangzhou 510000, China
liangzhaoheng23213@163.com, shenwuqiang@gdxx.csg.cn
[2] Guangdong Power Grid, Guangzhou 510000, China

Abstract. In order to improve the coverage of the target path corresponding to the generated software test data after optimization, and make the data better adapt to the software test process, the adaptive differential evolution algorithm is introduced to design the optimization method for software test data. Using the PSO algorithm to simulate the biological evolution mechanism in nature, and with the help of computer programming, the generated software test data are preliminarily trained; Draw on the path correlation based regression test data evolution of adaptive differential evolution algorithm to generate the relevant path representation method, mark the program to be tested, and construct the test data fitness function based on this; A hybrid model MPSO is proposed to select the best individual data in software test data; The selection criteria of scaling individuals are introduced into the adaptive scaling factor to cluster the optimal data individuals, so as to realize the design of optimization methods. The comparison experiment results show that the designed method has a good effect in practical application. This method can improve the target path coverage corresponding to the generated software test data on the basis of controlling the time length and evolution times required for software test data optimization.

Keywords: Adaptive differential evolution algorithm · Data selection · Individual clustering · Fitness function · Software test data · Optimization

1 Introduction

In order to ensure the quality of software, not only need advanced technical means, perfect R & D process, but also need continuous testing, no matter how advanced the current technology, how perfect the R & D process, can not 100% guarantee the software developed zero defects. Software testing can help scientific research and technology developers find out the errors or potential defects in the process of software development, and reduce the damage caused by software errors and potential defects as much as possible. In order to ensure the high reliability and integrity of software products, it is necessary to design enough test cases. However, the scale of software products is increasing day by day, the structure is becoming more and more complex, and the update

L. Yun et al. (Eds.): ADHIP 2023, LNICST 549, pp. 239–250, 2024.
https://doi.org/10.1007/978-3-031-50549-2_17

cycle is becoming shorter and shorter. It is a huge challenge for software testing to carry out comprehensive testing to find the problems and potential defects in the increasingly tight software development cycle. Therefore, it is necessary to use effective test cases within a limited test time to find out the errors and potential defects of the product as far as possible to ensure the accuracy and reliability of the software product, and the basis of this test is a high degree of target path coverage.

In recent years, the use of artificial intelligence algorithms such as ant colony algorithm, particle swarm optimization algorithm and genetic algorithm to automatically generate test cases has attracted more and more researchers' attention, and has achieved good research results. For example, Li Lu et al. [1] studied the defect optimization method of high-performance traffic analysis software based on particle swarm optimization algorithm, which introduced particle swarm optimization algorithm to analyze the current defect status of high-performance traffic analysis software, reset defect parameters, and build the defect optimization model of high-performance traffic analysis software. However, this method has the problem of long running time. Yang Bo et al. [2] studied a software fault location assisted test case generation method using improved genetic algorithm. Based on genetic algorithm, this method assisted the generation of test cases in the process of software fault location through the ranking of suspected faults in software fault location, but the marking path coverage of this method was low. The research shows that the adaptive differential evolution algorithm has the advantages of fast yield with fewer parameters and fast yield in high dimensions [3, 4]. Therefore, this paper takes the program code of unit test in software testing as the research object, introduces adaptive differential evolution algorithm to calculate all the tested paths, and uses the fitness function of the tested paths to improve the process of output optimal solution by optimizing the mutation operator, and improves the adaptive differential evolution algorithm by improving the mutation operator. Test cases covering each path are generated automatically to optimize software test data. This paper makes use of the convenient and efficient characteristics of artificial intelligence algorithm, which not only reduces the test cost, but also obtains a high efficiency of automatic case generation, moreover, the target path coverage of the generated software test data is more than 90%, which is more than 10% higher than that of the comparison method, which is of great significance to the improvement of product quality.

2 Software Test Data Generation

A lot of work has proved that PSO algorithm can solve the problem of software test data generation better than GA algorithm. Moreover, the PSO algorithm is simple, with fewer parameters to configure and easy to implement [5]. Therefore, this chapter introduces PSO algorithm to design the generation of software test data.

In this process, with the help of computer programming, the problem to be solved is expressed into strings (or chromosomes), that is, binary codes or digital strings, by using the PSO algorithm to simulate the biological evolution mechanism in nature, so as to form a group of strings, and they are placed in the problem solving environment. According to the principle of survival of the fittest, the strings that adapt to the environment are selected for replication, and the two gene operations are crossover and mutation, Create

a new generation of clusters that are more adaptable to the environment. After such continuous changes from generation to generation, finally converge to a string that is most suitable for the environment, and obtain the optimal solution of the problem [6]. That is, the training of test software requirement data is preliminarily realized.

On this basis, it should be clear that the essence of PSO algorithm is an optimization algorithm based on swarm intelligence, and its idea comes from artificial life and evolutionary computing theory. The algorithm simulates the behavior of birds flying to find food, and makes the group achieve the optimal goal [7] through the collective cooperation between birds. When using the PSO algorithm to generate software test data, it is necessary to input the software test data into the test environment first, and express the test environment as D, on D in the target retrieval space of dimension, each software test data is regarded as a particle, and the particle is represented as m, then m particles form a population, where the i particles in the d the position of the dimension is x_{id}, x_{id} the corresponding flight speed in space is v_{id}, the optimal location searched by this particle is p_{id}, the current maximum value of the entire particle swarm p_{gd} in this way, the particle position in space is updated, and this process is taken as the data update process, as shown in the following calculation formula.

$$v_{id}^{t+1} = wv_{id}^{t} + c_1 r_1 \left(p_{id} - x_{id}^{t}\right) + c_2 r_2 \left(p_{gd} - x_{id}^{t}\right) \tag{1}$$

In formula (1): v_{id}^{t+1} means on $t + 1$ the corresponding flight speed of particles in space at time; w is the inertial factor, w the larger the value of, the more suitable for large-scale exploration of the solution space, w the smaller the value of is, the more suitable it is to explore the solution space in a small range; v_{id}^{t} means on t the corresponding flight speed of particles in space at time; x_{id}^{t} means on t at the moment i particles in the d dimension position; c_1, c_2 represents normal number, called acceleration factor; r_1, r_2 represents a random number with a value between 0 and 1 [8]. On this basis x_{id}^{t+1} the calculation formula is as follows:

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{2}$$

On the basis of the above contents, it should be clear that, d the value range of is $1 \leq d \leq D$, for the d position of dimension x constrain the range of change and make it clear x the value range of is $[x_{d\,\min}, x_{d\,\max}]$, for the d dimensional velocity v constrain the range of change and make it clear v the value range of is $[v_{d\,\min}, v_{d\,\max}]$.

In the iteration, if the position and speed exceed the boundary range, the boundary value shall be taken. The particle position in the space can be updated according to Fig. 1 below. In this way, the spatial position of the software test data can be updated to generate more complete and global adaptive test data [9].

The termination condition of the PSO algorithm takes the maximum number of iterations or the predetermined minimum fitness threshold that the optimal position searched by the particle swarm optimization meets according to the specific problem.

Because p_{gd} it is the optimal position of the whole particle swarm. Therefore, the software test data can be generated according to the above steps, or it can be used as the retrieval process of global particles. The global PSO algorithm has a fast convergence speed, but sometimes it falls into a local optimum; The local PSO algorithm

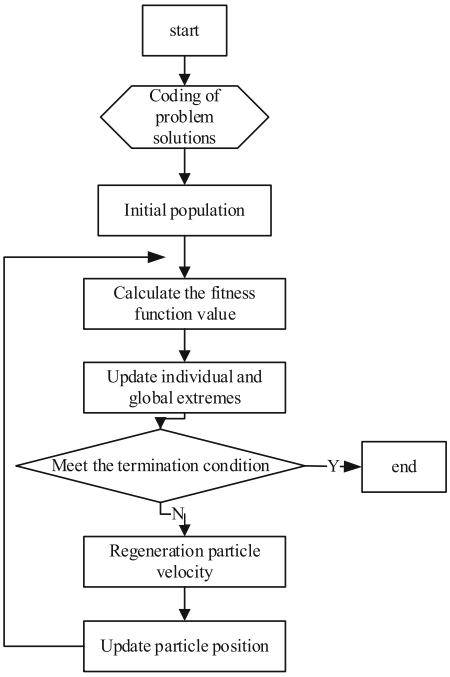

Fig. 1. Software test data generation and update process

converges slowly, but it is relatively difficult to fall into the local optimal value. Therefore, corresponding measures can be taken for data processing according to the specific requirements of software test data generation in practical applications.

3 Construct Test Data Fitness Function Based on Adaptive Differential Evolution Algorithm

On the basis of the above contents, in order to achieve the extraction of the optimal data in the generated software test data and master the fitness of different data in the generated data and the software test process, the adaptive differential evolution algorithm is introduced to construct and design the fitness function of the test data [10].

In this process, it should be clear that the establishment of the mathematical model of the multi-path coverage test data generation problem is closely related to the path representation method, and the tested program generally has multiple paths. In structured programming, the program usually consists of three structures: sequence, selection, and cycle. Among them, selection and cycle structure determine the trend of different branches of the program; For cyclic structure, by introducing Z path coverage can decompose it into multiple selection structures. The existing path representation methods include: using the statement number sequence of the program to represent the path, using the branch node sequence to represent the path, and using Huffman code to represent the target path. Using the path correlation based regression test data evolution generation

correlation path representation method in adaptive differential evolution algorithm for reference, the program to be tested is marked.

Assume that the program to be tested is Q, insert piles into each branch node in the program, and set the value of "true" to be 1 when crossing the corresponding branch node. If it is false, it is -1, and if it does not pass through the branch node, it is 0. Therefore, the code sequence of 1, -1, and 0 can be used to represent the path of program execution. Run the test program after inserting piles with a set of input data, and the branch node sequence is $D_1, D_2, \cdots, D_n$ the corresponding execution path is expressed as $q(T_i)$, where n is the path $q(T_i)$ length of. In addition, the program to be tested Q the target path of is $q_1, q_2 \cdots, q_N, N$ is the number of target paths.

Hypothetical procedure Q the input field of is α, input data is $T_1, \cdots, T_N$, then $T_N \in \alpha$, the path will be executed $q(T_i)$ Compare the coding sequence of and the target path from left to right, find the first different coding position, record the number of the same coding before, and record it as $\left|q(T_i) \cap q_j\right|$, take the ratio between it and the target path code, that is, the total length of the target path, to obtain the path similarity, and express it as $f_i(T)$, then $f_i(T)$ it can be calculated by the following formula.

$$f_i(T) = \frac{\left|q(T_i) \cap q_j\right|}{\max\left\{q(T_i), q_j\right\}} \tag{3}$$

In formula (3): $f_i(T)$ indicates the path similarity. After the above calculation is completed, the problem of solving test data generation can be transformed into solving $f_1(T), f_2(T), f_N(T)$ the fitness function of the maximum value problem is as follows.

$$H = \max(f_1(T), f_2(T), \cdots, f_N(T)) \tag{4}$$

In formula (4): H represents the fitness function of test data. According to the above method, the research on the construction of test data fitness function based on adaptive differential evolution algorithm is completed.

4 Selection of Optimal Individual Data in Software Test Data

After completing the above design, in order to select the best individual data in the software test data and seek the best test data, it should be clear that in the GPSO, particles evolve in the direction of the global optimal particles, and more obviously converge to the current optimal solution. Each time the particle position is updated, the characteristics of all particles in the population are integrated, and the information transmission speed is fast, but the population diversity is easy to lose, and the probability of falling into the local extreme value is large. In the local model LPSO, each particle only shares information with its neighbor nodes, which slows down the speed of information transmission, but the diversity of the population is guaranteed and it is not easy to fall into local extremum.

Therefore, based on the analysis of the impact of different fixed forms of neighbor patterns on the algorithm performance, it is found that the higher the average connectivity of social interactions between particle individuals, the faster the information transmission speed in the population, and the premature convergence phenomenon is more prone to

occur. In view of this, a hybrid model MPSO is proposed, that is, by observing the diversity index of particle swarm, each generation of particles selects GPSO or LPSO for evolutionary optimization, so as to realize the selection of the best individual data in software test data. In this process, describe the diversity of particles, that is, analyze the diversity of software test data in space. This process is shown in the following calculation formula.

$$I = \frac{1}{|k| \times |L|} \sum_{i>1}^{N} \sqrt{\sum_{d>1}^{D} \left(x_{id}^{t} - x_{d}^{t}\right)^{2}} \tag{5}$$

In formula (5): I represent the diversity of software test data in the space; k represents the size of particles in the search space; L indicates the maximum diagonal length in the search space. After calculation, output I the calculation results of I it represents the dispersion level of each particle in the community, I the larger the value, the more dispersed the population; I the smaller the value, the more concentrated the population, and the lower the diversity of the population.

Therefore, by observing the group I the feedback information automatically adjusts the topology of particle swarm to maintain the diversity of the community, maintain a certain population density, and avoid the algorithm falling into the local optimal value. On the basis of the above I the population with higher values will be globally and locally optimized, and the process is shown in the following calculation formula.

$$W(I) = \frac{I_i - l_{\min}}{e^{l_{\max} - l_{\min}}} \tag{6}$$

In formula (6): $W(I)$ represents the best individual data in software test data; $l_{\min}$ represents the minimum branch nesting depth; $l_{\max}$ represents the maximum nesting depth; e indicates the depth of the current branch. Complete the selection of the best individual data in the software test data according to the above method.

5 Optimal Individual Clustering and Optimization of Software Test Data Set

On the basis of the above contents, extract the optimal individual data from the software test data selected in the above way, introduce the selection criteria of scaling individuals into the adaptive scaling factor, and cluster the optimal data individuals. The process is shown in the following calculation formula:

$$r = \sum_{i>1}^{n} M r_i / \sum_{i>1}^{n} r_n \tag{7}$$

In formula (7): r represents the optimal individual clustering; M represents the cluster center. After completing the above calculation, the adaptive differential evolution algorithm is applied to automatically generate the model of test data, as shown in Fig. 2 below.

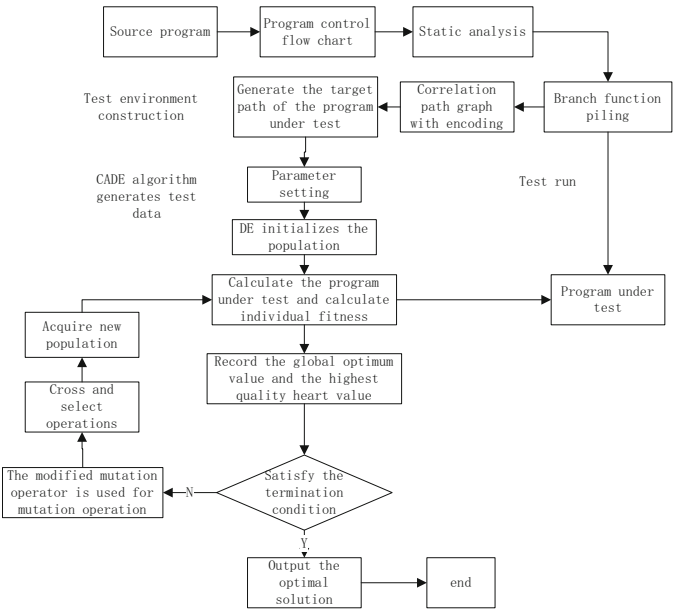

Fig. 2. Optimal solution of output software test data

The construction of the test environment is the basis for the generation of test data. In this stage, the source program is statically analyzed, and the coded test program flowchart generated by branch function instrumentation is used to generate the target path. This method is more concise than the traditional data flowchart.

However, as far as the current research is concerned, the existing methods assume that the mass of each particle in the population is the same, and do not take into account the differences between the particles. Therefore, in the optimization process, the process of outputting the optimal solution is improved. On the basis of the existing algorithms, a mutation operator based on the center of mass is proposed. The improved mutation operator can be calculated by the following formula.

$$A(t) = x_i(t) + F(r + c_1 r_1 + c_2 r_2) \tag{8}$$

In formula (8): $A(t)$ represents the improved mutation operator; F represents the mass of individual particles. After the selection of the improved mutation operator is completed, in order to break the stagnation of evolution, jump out of the local best, let the operator fluctuate in size, and by limiting its value range, make the difference dynamic scaling, and search for optimization in a large range, theoretically it can improve the ability of global optimization. In the early stage of evolution, the operator tries to maintain the diversity of the population, and takes the small value first; in the later stage, the drill takes the large value to speed up the convergence speed. In this paper, the corresponding generation operator of normal distribution random number is used as follows:

$$\beta = normrnd(U \times F) \cdot A(t) \tag{9}$$

In formula (9): β represents the generating operator process; *normrnd* represents the deviation compensation value; U represents a normally distributed random number. By compensating the deviation of the operator generation process, the software test data can be better optimized within the optimal range, which is helpful to jump out of the region where the local extreme value is located quickly.

The cross factor plays a fine tuning role, and the appropriate value can maintain the diversity of the population. If the crossover probability decreases, the candidate will contain more target individuals. If the crossover probability increases, the weight of variant individuals in the newly generated candidate will increase.

At the early stage of the iteration, the diversity of the population is relatively high, so cross operation with a small probability can maintain the diversity of the population. As the number of iterations increases, the diversity of the population is gradually losing and approaching the extreme point. Therefore, local search should be increased to accelerate the convergence of the algorithm, so as to approach the extreme point at a faster speed.

In this process, it should be clear that generating test data is the core part of the whole model based on the adaptive differential evolution algorithm. In this stage, the population is initialized according to the target path and test data range constructed by the test environment, combined with the setting of parameters required by the algorithm, and the population is updated by running the improved mutation operator to guide the population to evolve towards the optimal value; And use the test data to run the program to be tested after pile insertion to calculate the fitness value. When the fitness value meets the needs of software testing, complete the data optimization design.

According to the above method, complete the optimal individual clustering and software test data set optimization, and realize the design and research of software test data optimization method based on adaptive differential evolution algorithm.

6 Comparison Experiment

The research on optimizing software test data using adaptive differential evolution algorithm has been completed from four aspects above. In order to test the application effect of software test data optimization methods, the following will take the tested software program provided by a teaching and research institute as an example to design a comparison experiment and carry out the research as shown below.

In order to ensure that the designed method can play its expected role in the test, the environment in which the software program under test runs is described before the experiment, and the relevant contents are shown in Table 1 below.

According to the above contents, after completing the design of the technical parameters of the comparative experimental environment, the method designed in this paper is used to optimize the data of the tested software. In the optimization process, it assists modern intelligent algorithms such as GA to generate software test data. On this basis, the adaptive differential evolution algorithm is introduced to construct the fitness function of the test data. In order to ensure that the constructed fitness function can be used as the basis for evaluating the software test data, the parameters of the algorithm in the computer can be set according to Table 2 below during the iteration of the adaptive differential evolution algorithm.

Table 1. Technical Parameters of Comparative Experimental Environment

S/N	project	parameter
1	Experimental programming language	Java Language
2	Experimental program running environment	Eclipse environment
3	Microcomputer environment	Windows operating system Intel (R) Core (TM) i3 CPU 2.0 GHz
4	Computer running memory	2 GB RAM

Table 2. Training parameter settings of adaptive differential evolution algorithm

S/N	project	parameter
1	Population iterations (times)	1000
2	Operator crossover probability (%)	0.6
3	Operator compilation probability (%)	0.01
4	Number of independent runs for each case of different programs (times)	50
5	Data cycle branch (s)	6
6	Select nesting quantity (pcs)	3
7	search space	3
8	Learning factor	1.5
9	Inertia weight	0.9
10	Diversity threshold in algorithm	0.1

After setting the experimental parameters, select the best individual data in the software test data, and optimize the software test data set by clustering the best individual data.

On the basis of the above content, software test data optimization method based on particle swarm optimization (reference [1] method) and software test data optimization method based on genetic algorithm (reference [2] method) are introduced, and the introduced method is regarded as traditional method 1 and traditional method 2. The method in this paper and the two traditional methods are used to optimize software test data.

In the optimization process, three software test data populations of different scales are selected, and three methods are used to evolve the software test data respectively. The average length of time required by the three methods to optimize the software test data is compared, which is the key basis for testing the application effect of the method in this paper. The statistical experimental results are shown in Table 3 below.

From the experimental results shown in Table 3 above, it can be seen that the average time required for optimization of software test data using this method is less than 1 s,

Table 3. Average time required for three different methods to optimize software test data

S/N	Population size	Average duration (s)		
		Methods in this paper	Traditional method 1	Traditional method 2
1	50	0.125	0.569	0.639
2	100	0.156	0.693	0.678
3	150	0.269	0.854	1.069
4	200	0.489	0.910	1.598
5	300	0.526	1.025	1.756
6	400	0.963	1.156	1.963
7	500	0.956	1.569	3.051

while the average time required for optimization of software test data using traditional methods is significantly higher than the average time required for optimization of data using this method.

After completing the above experiment, set the input range of software test data to [0, 64], [0512], [1024], use three methods to optimize the software test data, and compare the average evolution times of the three methods on the input data during the optimization process. The results are shown in the following figure:

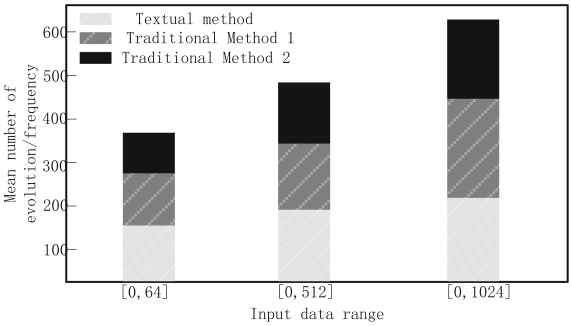

Fig. 3. Average evolution times of three methods for input data in different ranges during optimization

It can be seen from Fig. 3 above that the three methods optimize the training of input data in different ranges. With the increase of input data range, the average evolution times of the three methods show an increasing trend. Use this method to optimize [1024] software test data, and the average number of evolutions in the optimization process is less than 300; The traditional method 1 is used to optimize software test data, and the average evolution times are less than 500; The traditional method 2 is used to optimize the software test data, and the average evolution times are less than 700.Based on the above results, it can be seen that among the three methods, only using this method to optimize software test data can ensure that the average evolution times are at a relatively

low level, while using traditional method 1 and traditional method 2 to optimize software test data, the corresponding method has more average evolution times, that is, the data optimization process is more complex.

After the above design is completed, several software test paths are set according to the software test requirements. It is known that different software test paths require different software test data sizes and categories. After comparing the three methods to optimize the software test data, the generated test data can cover the extent of the target test path, that is, compare the availability of the optimized corresponding data in the software testing process. According to the above method, carry out a comparison experiment and make statistics of the experimental results, as shown in Fig. 4 below.

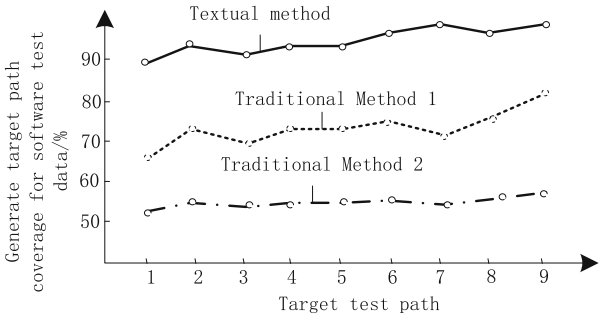

Fig. 4. Target path coverage corresponding to software test data generated by the three optimized methods (%)

From the experimental results shown in Fig. 4 above, it can be seen that the method in this paper is used to optimize the software test data, and the target path coverage corresponding to the generated software test data after optimization is >90%, while the target path coverage corresponding to the generated software test data after optimization in the traditional method 1 is between 60% and 80%; The target path coverage corresponding to the software test data generated after optimization of traditional method 2 is between 50% and 60%.

Based on the above experimental results, the following experimental conclusions are obtained: compared with the traditional methods, the software test data optimization method designed in this paper using adaptive differential evolution algorithm has a good effect in practical application. This method can improve the target path coverage corresponding to the generated software test data on the basis of controlling the time required for software test data optimization and the number of evolutions. In this way, it provides further support and guidance for the standardized implementation of software program testing and other related scientific research work.

7 Conclusion

In order to find potential software defects as much as possible and improve the coverage of target paths corresponding to software test data generation, this paper conducts testing research on the optimization method of software test data through software test

data generation, construction of test data fitness function based on adaptive differential evolution algorithm, selection of optimal individual data in software test data, optimal individual clustering and optimization of software test data set. After completing the design of this method, taking the software program under test provided by a teaching and research unit as an example, a comparative experiment was designed to prove that the method designed in this paper has a good effect in practical application. This method can improve the coverage of the target path corresponding to the generated software test data on the basis of controlling the length of time required for software test data optimization and the number of improvements.

References

1. Li, L.: Defect optimization method of high-performance traffic analysis software based on particle swarm optimization. J. Anhui Techn. Coll. Water Resour. Hydroelectr. Power **21**(4), 56–59 (2021)
2. Yang, B., He, Y., Xu, F., et al.: IGA: software fault location assisted test case generation method using improved genetic algorithm. J. Beijing Univ. Aeronaut. Astronaut. **48**(3), 1–13 (2022)
3. Zhang, X., Liu, Q., Qu, Y.: An adaptive differential evolution algorithm with population size reduction strategy for unconstrained optimization problem. Appl. Soft Comput. J. **138** (2023)
4. Niu, D., Liu, X., Tong, Y.: Operation optimization of circulating cooling water system based on adaptive differential evolution algorithm. Int. J. Comput. Intell. Syst. **16**(1) (2023)
5. Li, W., Sun, Y., Huang, Y., et al.: An adaptive differential evolution algorithm using fitness distance correlation and neighbourhood-based mutation strategy. Connect. Sci. **34**(1) (2022)
6. Gouda, S.K., Mehta, A.K.: A self-adaptive differential evolution using a new adaption based operator for software cost estimation. J. Inst. Eng. (India) Ser. B **104**(1) (2022)
7. Chen, J., Chen, X., Zan, T., et al.: An automatic generation of software test data based on improved Markov model. Web Intell. **20**(4) (2022)
8. Ferreira Vilela, R.,Choma Neto, J., Santiago Costa Pinto Victor, H., Lopes de Souza, P.S., do Rocio Senger de Souza, S.: Bio-inspired optimization to support the test data generation of concurrent software. Concurr. Comput. Pract. Exper. **35**(2) (2022)
9. Mohaideen Abdul Kadhar, K., Narayanan, N., Vasudevan, M., et al.: Parameter evaluation of a nonlinear Muskingum model using a constrained self-adaptive differential evolution algorithm. Water Pract. Technol. **17**(11) (2022)
10. Esnaashari, M., Damia, A.H.: Automation of software test data generation using genetic algorithm and reinforcement learning. Expert Syst. Appl. **183** (2021)