Journal of Artificial Intelligence and Data Mining (JAIDM), Vol. 9, No. 4, 2021, 465-474.

Shahrood University of Technology

Journal of Artificial Intelligence and Data Mining (JAIDM)
Journal homepage: http://jad.shahroodut.ac.ir

Research paper

Software Testing using an Adaptive Genetic Algorithm

Amirhossein Damia[1*], Mehdi Esnaashari[2] and Mohammadreza Parvizimosaed[3]

1,3. Department of Computer Engineering, K. N. Toosi University of Technology, Tehran, Iran.
2. Faculty of Computer Engineering, K. N. Toosi University of Technology, Tehran, Iran.

Article Info

Article History:
Received 01 September 2020
Revised 07 May 2021
Accepted 14 July 2021

DOI:10.22044/JADM.2021.10018.2138

Keywords:
Software Test, Test Data Generation, Path Coverage, Search Algorithms, Genetic Algorithm.

**Corresponding author: damiaa@email.kntu.ac.ir(A.H. Damia).*

Abstract

In the structural software test, the test data generation is essential. The problem of generating the test data is a search problem, and for solving the problem, the search algorithms can be used. Genetic algorithm is one of the most widely used algorithms in this field. Adjusting the genetic algorithm parameters helps to increase the effectiveness of this algorithm. In this paper, the adaptive genetic algorithm is used in order to maintain the diversity of the population to the test data generation based on the path coverage criterion, which calculates the rate of recombination and mutation with the similarity between the chromosomes and the amount of chromosome fitness during and around each algorithm. The experiments performed show that this method is faster for generating the test data than the other versions of genetic algorithm used by the others.

1. Introduction

Software testing is a process of identifying the correctness of software by considering its all attributes (reliability, scalability, portability, re-usability, usability) and evaluating the execution of the software components to find the software bugs or errors or defects. Software testing provides an independent view and objective of the software, and gives surety of the software's fitness. It involves testing all the components under the required services to confirm that whether it is satisfying the specified requirements or not. The process also provides the client with information about the quality of the software. Once the software is produced, it must be tested. According to the research work conducted by NIST, the damage caused by software breaches is massive [43].

The software test consumes many resources but does not add any new functionality to the product. Therefore, significant efforts have been made in order to reduce the software development costs by developing the automated software testing tools. In the last decade, various methods have been introduced for automatic software testing in order to maximize error detection by producing the least number of test data. In the process of test data generation, we require a criterion such as a path coverage criterion, the edge coverage criterion, and the node coverage criterion to determine the amount of program coverage by the data generated. After selecting the quality criterion, the amount of program coverage can be determined by the generated data. The automated structural testing methods are divided into two categories: dynamic and static methods [2]. The generation test data in static structural is based on the analysis of the program's internal structure without the need to run the program, and is usually based on the symbolic execution. This method has problems with the arrays and pointers [1].

In the dynamic method, the program under test (PUT) is required to be run, and the problems of the static method are solved. In this method, generating the test data is transformed into an optimization problem to use the search algorithms. In other words, parts of the test program can be overwritten as a function, and searched for the optimal value of the resulting function. The data obtained covers the intended purpose according to the quality criterion.

In generating the test data, we require a criterion referred to as the coverage criterion in order to determine the amount of program coverage by the generated data. These criteria are defined based on the control flow graph (CFG) of the program. For example, the branch coverage or edge coverage criterion is one of the criteria in which the goal is to cover all the branches of the program CFG. Another quality criterion is the statement coverage or node coverage criterion, which aims to execute all nodes of CFG. Another criterion is the path coverage criterion, which aims to cover all paths within CFG. The branch coverage and node coverage are subsets of the path coverage criteria [44]. After selecting a coverage criterion, it is possible to determine the program coverage based on the selected criterion using the generated test data.

Many previous dynamic methods have used meta-heuristic or evolutionary algorithms such as simulated annealing, genetic algorithm, and particle swarm optimization [3,6]. Genetic algorithm is one of the widely used algorithms in the field of automation of software test data generation [21-24].

The main problems with these methods are scalability, premature convergence, and slow speed. Scalability involves the size of the software under the test, which could be addressed with the method. Xiao *et al*. [45, 46] have reported that the genetic algorithm (GA)-based approaches can handle a larger number of branches with large search spaces. However, they suffer from the problem of slow convergence rate. This is mainly due to the numerous parameters such as the crossover rate (Pc) and mutation rate (Pm), which must be tuned. In addition, attempts to speed up GA usually results in premature convergence to the local optima. The proposed method in this paper tries to tune these parameters adaptively in such a way that an acceptably good solution is found with few number of evaluations, i.e. the proposed method both speeds up GA and prevents it from a premature convergence.

Many researchers have proposed the different p_c and Pm strategies. They can be categorized as constant, random, deterministic, and adaptive strategies [46, 50]. In constant to the Pc and Pm strategies, if the value of *Pc* is too high, it will cause the chromosome to lose its ability to adapt, and if it is too small, the number of children produced will not be enough. In mutation, if the rate of this operator is high, the algorithm acts as a random search, and if the rate of this operator is low, the algorithm may be caught in local optimizations, which can delay the convergence. The random *Pc* and Pm strategies increase the convergence of GA in an early stage of the algorithm. In the time-varying Pc and Pm strategies, Pc and Pm are defined as a function of time. However, all these strategies have the weakness of a premature convergence to the local minimum. In order to overcome this weakness, the adaptive Pc and Pm strategies have been proposed in this method, and the search status is taken during the execution of the feedback algorithm. It is used in order to improve the searching capability, and to maintain the diversity of the population by adjusting Pc and Pm.

GA has difficulties in giving stable results (stuck up at local optima); the convergence is slow and has a non-explicit memorization of the best individuals. In order to overcome this problem, in this paper, one adjusts the cross-over rate and mutation rate by maintaining the population diversity. This paper employs measures of the population diversity in order to adapt cross-over and mutation rate: standard population diversity (SPD). SPD is pure to the solution space diversity with no regard to the health/fitness of the individuals [4].

In general, in this method, the chromosomes that have a better fitness and diversity than the other members of the population are considered to have a lower rate of mutation and recombination rate, and in contrast, the chromosomes that have less fitness and diversity have a higher rate of mutation and recombination. In order to evaluate the efficiency of the proposed method, this method is used to automatically generate the test data on many different programs, and the results obtained are compared with the results the other versions of GA and the other work done by the others. The results obtained show the obvious superiority of the proposed method.

2. Related Works

Much research work has been done on automating the generation of an efficient test data. The following are some of these studies and their achievements.

GA has been very successful. An improved GA optimization has been proposed to overcome the traditional controller needs like stability and control speed [34, 19-24]. The hybrid of GA and asexual reproduction optimization is used to impute the missing values [35]. A predictive model in polymers has been designed using GA [36]. Another work has implemented GA in order to find the unpredicted thermal conductivity improvement in disarranged nano-porous graphene [37]. A method has used GA integer-valued optimization to improve the machine learning models' feature configuration and architecture [38]. A model has been proposed based on an adaptive genetic algorithm with fuzzy logic (AGAFL) in order to predict heart disorder [41]. GA is used for imputing the missing values to predict the hospital length od stay using the NICU datasets [49, 50]. GA has been implemented on an IoT platform for the customer needs [52, 53].

GA in the software test has been able to succeed in both the parameter and structural optimizations [5]. GA [6-8] and the combination of GA with other algorithms are the most common methods that the researchers have used in the recent years [9-11, 53, 55]. Gupta and Gupta have focused on the use of GA for generating the test data that can cover the most error-prone path so that emphasis can be given to test these paths first [12]. Suresh has used GA to generate the test data for feasible basis paths. Their results showed that GA was more effective and efficient than a random testing method [13]. A novel method has been designed methodology for the test data generation using GA to cover the most critical path of a program [14].

Singh has used GA to automate the generated test data based on the path coverage criteria. Their results showed that the quality of the test data was higher than the quality of the test data generation at random [15].

A novel method of particle swarm optimization (PSO) algorithm has been proposed to generate the test data automatically [51].

A method has been presented for path testing by automatically generating the test data and optimizing the test data to test the critical paths for the software under the test using a real-coded GA. In the proposed approach, a one-to-one injective mapping scheme is used to map the test data to the corresponding path, and the most critical path is covered during path testing of a specific software. The proposed method can reduce the number of test data generation required for path testing, and is faster than the traditional GA in covering a critical path [28].

GA is used to achieve both the path and branch coverage of the program in the test data generation [16]. In another method for test data generation, a dynamic test is based on the PSO algorithm used. The performance of this method was better than the random search and tabu search [17].

In [18], a method for test data generation has been performed using GA. The efficiency of the proposed method is based on the dependence of the program data and in comparison with the random search method, the results of which showed that their proposed method is better than the random search. Mack Mann and Pradeep Tamar have introduced a GA-based method for generating software test data, and their results were compared with the stochastic method. In this paper, the impact of the early population on the efficiency of GA is investigated. Their experiments showed that their proposed method was more efficient than the random method, and required less time to generate the software test data, and by increasing the initial population size, more search space could be created by increasing diversity, making it a less likely algorithm [25, 26].

Sahoo *et al*. have proposed a method using a PSO algorithm to cover the critical paths in CFG. It is called a critical path if the probability of its coverage is low. One of the well-known fitness functions for the test data generation problems is the branch distance and approach level function, which has been used in many works. In this paper, the problems related to this function are investigated, and instead of the approach level, the path distance is used [27].

Manikumar *et al*. have presented an incremental GA for branch coverage testing. Initially, a classical GA is used to construct the population with the best parents. The incremental GA starts with these parents as the initial population. This work aims to solve the problem of a large population. Hence, it is unnecessary to maintain a huge population size and many iterations to cover all the branches. The experimental results obtained indicate that the proposed incremental GA search technique outperforms the other meta-heuristic search techniques in memory usage and scalability [47]. Kumar *et al*. have proposed an approach to automatically generate the test data for data flow testing based on a hybrid adaptive PSO-GA algorithm. The hybrid APSO-GA is developed to conquer the weaknesses of the GA and PSO algorithms, especially in data flow testing. The results obtained show that hybrid adaptive PSO-GA gives better results as compared to the other algorithms that are used in the field of test data generation [48].

Mishra *et al*. have presented a method for path testing by automatically generating the test data and optimizing the test data to test the critical paths for software under the test using a real-coded GA. In the proposed approach, a one-to-one injective mapping scheme is used to map the test data to the corresponding path, and the most critical path is covered during path testing of a specific software. The proposed method can reduce the number of test data generation required for path testing, and is faster than the traditional GA in covering a critical path [28].

One of the main problems in the firefly algorithm is getting stuck in the local minima, and consequently, a poor exploration. Poor exploration and population diversity are directly related. In order to solve this problem, Damia *et al*. have used the asexual reproduction optimization algorithm in the structure of the firefly algorithm to diversify the population of this algorithm. The results of their experiments show that their proposed hybrid method is better than each one of these algorithms [7-9].

In [13], the ant colony optimization algorithm is used to generate the test data. Their proposed method is an evolutionary strategy to improve the search performance of ants in local movements and increase the exploitation of the search. The results of their experiments show that their proposed method is more efficient than the existing test data generation techniques in terms of branch coverage and convergence speed.

3. Problem Formulation

The path test method executes all the control path method paths under test at least once in order to execute all the program commands. To perform this test, the current control structure of the method under the test is considered as a control flow graph (CFG).

In this paper, the problem is to automatically generate the test cases for a given software under the test (SUT). The coverage criterion is considered to be the path coverage, i.e. a perfect test method has to traverse every execution path within SUT. In order to perform this, first, the control structure of SUT is transferred into a control flow graph (CFG). For example, CFG of the following SUT is shown in Fig. 1.

```
1: void main ()
2:  {
    int x ;
    x = scanf (“d”);
3:    if (x > 0)
4:         x++
5:    else if (x <= 0)
6:         x - -
7:    print (x)
8:   }
```

The problem here is to automatically determine the set of inputs to test a specific method that can cover all paths. For this purpose, a search on the input parameters of the method is used. A method is generally defined as [return type] method_name ([input parameter list]). Next, the cyclomatic complexity of CFG is determined [54, 55].

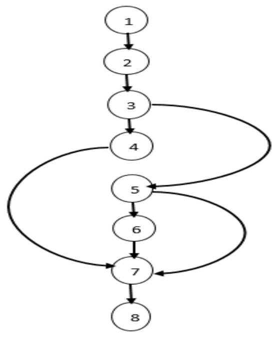

Figure 1. An example program and its CFG.

The Cyclomatic complexity is equal to the number of linearly independent paths within the control structure of SUT. As a result, a perfect path test method has to find a test set that is able to cover all of these linearly independent paths. In order to be able to generate such a test set, a comprehensive search over the input parameter space of SUT is used. Note that a SUT is generally defined as:

[return type] method_name ([input parameter list])

Thus the input space of SUT is the Cartesian product of its input parameter list. For example, if the method has two input parameters, the input space will be a pair $<i_1, i_2>$, where i_1 and i_2 indicate the values assigned to the first and the second parameters, respectively. The main goal of the search is to minimize the number of calculations required to find the suitable test set. This goal is important because a typical program usually consists of many SUTs. Therefore, the amount of time and calculations required for generating a suitable test set for a single SUT within the program, significantly affect the overall time and processing needs of generating the test sets for the whole program.

The main goal of the search work is to minimize the number of calculations required to identify the inputs because as the program grows and the number of functions in it increases, it will take a long time to generate these test items, and the faster the automatic solution with fewer calculations, it would be better to cover the paths.

4. Proposed Method

In this work, we used GA with its tuned parameters for testing the software. Early convergence is one of the significant problems in GA, and there is a direct relationship between early convergence and lack of population diversity. A population is diverse if the distance between its chromosomes is large; otherwise, it is small. The similarity between the chromosomes can be calculated based on the distance between them.

4.1. Calculating SPD

SPD is calculated by finding the site of the mean chromosome within the population according to Equation (1) and the sum of inputs (genes) Euclidean distances from this mean point to the site of each chromosome according to Equation (2) [4].

$$G_n^{avg} = \frac{1}{P}\sum_{i=1}^{P} G_{i,n} \qquad (1)$$

$$SPD_i = \sqrt{\sum_{n=1}^{N} (G_{i,n} - G_n^{avg})^2} \qquad (2)$$

The population size is $(G1toGP)$ where each chromosome consists of N gene. Gin is the nth gene of chromosome i; $Gi = (Gi,1, Gi,2,..., Gi,N)$.The mean chromosome in the population is G^{mean}, and is calculated as the gene mean overall ps chromosomes. G_n^{mean} is the mean of all nth genes in the population. $SPDi$ is the chromosome $I's$ the portion to SPD. It is calculated as the Euclidean distances between chromosome i and G^{mean} [4].

$SPDi$ can be used to determine SPD. In order to calculate SPD, the standard deviation of the population is calculated aaccording to Equation (3) $\sigma(G_n^{mean})$, and SPD is calculated according to Equation (3) [4].

$$\sigma(G_n^{avg}) = \sqrt{\frac{1}{P} + \sum_{i=1}^{P} (G_{i,n} - G_n^{avg})^2} \qquad (3)$$

$$SPD = C_v(G^{avg}) = \frac{1}{N}\sum_{j=1}^{N} \frac{\sigma(G_j^{avg})}{G_j^{avg}} \qquad (4)$$

4.2. Adaptive Cross-over

This operator is applied to a pair of chromosomes and in the form of different are introduced, the most important of which are one-point, two-point, multi-point, and uniform. This operator is set based on a probability of Pc. If the probability of Pc is high, the good chromosomes may be easily damaged, and if this probability is low, the new chromosomes may not be formed. Therefore, it is better to calculate Pc based on the fitness of each chromosome during the search [4]. The proposed method for calculating Pc is according to Equation (5) [4]. In this work, the uniform cross-over is used.

$$Pc = [(\frac{SPD}{SPD_{\max}} * (k2 - k1) + k1] \qquad (5)$$

In this work, *Pc* is in the range of 0.5 and 0.85.

4.3. Adaptive Mutation

The purpose of this mutation operator is to escape the algorithm from local optimization and maintain the population diversity. This operator occurs based on a probability *Pm*.

If the value of probability *Pm* is high, the algorithm acts as (2) a random search, and if this value is low, the algorithm gets stuck in the local minimum so a suitable method is to parameterize this probability [4].

Therefore, the adaptive mutation is used in this work, which is calculated to be a probability of Pm while executing the algorithm.

The adaptive mutation is combined with the following two methods [4]:

- Impact of diversity
- Impact of fitness

Equation (6) proposes the impact of diversity $p_m^{diversity}$ and Equation (7) proposes the impact of fitness $p_m^{Fitness}$ [4].

$$p_m^{Diversity} = \frac{SPD\max - SPD}{SPD\max} * k \qquad (6)$$

$$p_m^{Fitness} = k * (\frac{f\max - f}{f\max - f\min}) \qquad (7)$$

In Equation (7), f is the parent fitness, $f\max$, and $f\min$ are the best and worst fitness chromosomes in the population, respectively (*k is 0.5*).

Equation (8) defines the proposed method for calculating Pm for each chromosome [4].

$$Pm = \frac{P_m^{Fitness} + P_m^{Diversity}}{2} \qquad (8)$$

4.4. Population Initialization

After generating the CFG program, we then obtain the paths of this graph.

In order to start searching throughout the state space of the problem, a random population is initially generated. Each chromosome in this population represents a test set for PUT. Suppose the number of input parameters of the PUT to be Nv and the number of finite paths within the corresponding CFG to be Np.Then each chromosome t_i would be of the form $ti = [ti(1), ti(2), ..., ti(Np)]$, where each $t_i(b)$ represents a test data for evaluating PUT with a specific set of input parameters. Let $ti(b) = [\text{xib}(1), \text{xib}(2), ..., \text{xib}(\text{Nv})]$. Here, $x_{ib}(l)$ specifies the lth input parameter for PUT in the test data $t_i(b)$.

In this work, chromosome representation is integers.

4.5. Fitness Function

After generating the initial random population, in order to determine the fitness of each population member, it is necessary to run the test method N_p times to determine which paths are covered by the input test set. The fitness of each chromosome is calculated according to Equation (9).

$$fitness(chromosome)_i = \frac{SatsifiedPath(i)}{Np} \quad (9)$$

4.6. Selection

The selection operator is used to determine the chromosomes to be used as the parents in the creation of the off-spring that population the subsequent generation. The rank-based selection method is one of the widely used in GA [31] [32] [33].

In the Rank-based selection method, instead of using the absolute fitness value, the fitness rank of the members in the population is used to determine the selection. In this method, the fitness of the best member of the population equal to ns is considered. The second-best member of the population is assigned a fitness of $ns-1$, and this continues until the weakest member of the population is reached (it is a debt that the fitness of the weakest member will be equal to 1). Note that in a common GA, if the fitness of two members in the population is the same, we must randomly attribute one fitness R and the other fitness $R-1$. In the proposed method, if the fitness of the two members is equal, their rank will be determined based on the diversity of each member.

4.7. Stopping condition

The termination conditions in AGA specify the stopping criteria after the desired solution is obtained in few numbers of iterations. The termination condition in AGA can occur due to the following reasons:

- A finite number of generations (in this paper is 100000).
- The optimized solution is obtained.
- Implementation of algorithm
- The implementation of the pseudo-code of improved genetic algorithm (IGA) is given in Algorithm 1.

Algorithm 1. Test data generation based on improved genetic algorithm (IGA).

```
1: Input: instrumented version of a program to be tested
2:        number of variants program under test (NV)
3:        number of paths program under test (NP)
4:        max iteration
5:        ps
6: Output: set of test data
7: Begin
8: chromosome size = NP * NV
9: population = GenerateRandomSolutions(population size, chromosome size)
10: iteration = 0
11:   while iteration < max iteration  do
12:        Evaluate(population)
13:        selection result = SelectParents ( )
14:        Update $p_c$ according to Equation (5)
15:        Recombination ( )
16:        Update $p_m$ according to Equation (8)
17:        Mutation ( )
18:        population = new population
19:        iteration = iteration + 1
20:   end while
21: return output
22: End
```

5. Experiment

This experiment aims to show the superiority of the proposed method compared with the works done by the others in Table 1. Each algorithm was executed 50 times. For each execution, the algorithms were performed with the same range of input variables; these programs are listed in Table 2.

The decision for the termination criteria is that if at least one test datum has been found to traverse the paths or the number of iterations of the evolution is reached the present value (maximum iteration), the evolution will stop.

The decision for the termination criteria is that if at least one test datum has been found to traverse the target path or the number of iterations of the evolution reaches the present value, the evolution will stop. The evaluation criteria to test the effectiveness of different methods are listed as follow:

Evals: Number of evaluations for individual evaluation of each method.

In order to ensure that the number of chromosomes has no effect on the performance of the compared methods, all methods have adopted the same population size and the same initial population. Each experiment was repeated 50 times, and the results obtained were reported as the average of all repetitions. The comparison results are given in Tables 4, 5, and 6. In all tables, the mean, standard deviation, P-value (t-test with α = .05), and percentage of covered paths are summarized for each algorithm per benchmark program. The results obtained confirm that the proposed IGA method outperforms the other existing state-of-the-art methods in terms of the number of fitness evaluations. The main reason for the superiority of IGA over the existing algorithms is its ability to escape from the local optima. This ability is due to the suitable setting of recombination and mutation rates considering the fitness of each chromosome and its degree of diversity in the population. In [39], [40], [47], [1], and [28], these rates are constant, and thus no feedback is available from the search space. Therefore, the algorithm is easily trapped into the local optima. This problem is less severe in [10,11], where a combination of GA and PSO are utilized. However, the method is more complex, and thus takes much more time to achieve suitable results. The method of [29] places great emphasis on the population diversity. However, increasing the population diversity leads to a slower convergence speed. The problem with the method given in [4] is that even though it varies the values of its parameters in time, it considers no feedback from the search space for adjusting its parameter variation scheme.

6. Conclusions and Future Work

In this paper, we proposed an automatic test data generation method based on an adaptive genetic algorithm. The method improves the search efficiency by maintaining the population diversity. The experimental results obtained show that the proposed method is more effective than the existing similar to path testing. Although the subjects selected in this work are Python language, the thought of this method can be used for reference in other languages as the experimental objects. For the future work, we will use this method for the object-oriented programming and classes.

Table 1 shows the parameters of genetic algorithm used in the previous works. The value of each parameter varies based on the nature of the datasets that are used. The cross-over rate is generally more than 0.5, and the mutation rate is between 0.01 and 0.15. Different selection algorithms can also be used for each work.

Table 1. Work of others.

Algorithm	Cross-over rate	Mutation rate	Selection
Suresh [39]	0.5	0.05	Elitism
Shimin [40]	0.8	0.01	Ranking
Ghiduk [42]	0.8	0.15	Roulette wheel
Manikumar [47]	1.0	0.01	Tournament
Mishra [28]	0.8	0.02, 0.03, 0.07	Ranking
Bao [29]	Adaptive	Adaptive	Roulette wheel

The value of each parameter was set based on the experiment in Table 2. The range of the input variables was between -50 and 50. The maximum iteration of the algorithm was 100000, and ps was 30.

Table 2. Selected programs for experiments.

PUT	Description
Triangle classification [29]	Find the type of triangle
Fibonacci [30]	Find Fibonacci sequence
Quadratic equation [41]	Equation of the second degree

The algorithms apply the same values of parameters, which are listed in Table 3.

Table 3. Parameters of algorithm.

Parameter	Value
p_s	30
Maximum iteration (stop condition)	100000
Range of input variables (it is a search space and is an option that can be defined in any interval according to the user's needs)	[-50, 50]

Table 4 compares the proposed method with the other methods based on the mean, std, ttest, pvalue, and percentage path coverage. The pvalue of AGA is higher than the other methods. The ttest of AGA is 0, while the average of other methods ttest is 3. (Tables 5 and 6 are like Table 4 but I do not have any idea about the title of them. Please write the details about the other two tables.

Table 4. Triangle classification program.

Algorithm	Mean	std	ttest	pvalue	Percentage path coverage
Suresh [39]	21245.2	20528.1	5.131	0.0	100 %
Shimin [40]	13651.6	17205.6	3.078	0.002	100 %
Ghiduk [42]	10510.6	9727.0	3.047	0.002	100 %
Manikumar [47]	15294.4	13297.5	3.483	0.0	100%
Kumar [10]	6481.0	5743.9	0.444	0.657	100%
Mishra [28]	16649.8	12359.2	5.708	0.0	100%
Sahoo [27]	12923.4	10094.1	2.002	0.048	100%
Bao [29]	7140.4	6081.2	1.022	0.308	100%
AGA	5806.7	4701.9	0.0	1.0	100 %

Table 5. Quadratic equation.

Algorithm	Mean	std	ttest	pvalue	Percentage path coverage
Suresh [39]	6897.98	5395.9	3.243	0.001	100 %
Shimin [40]	9603.36	8974.9	4.123	0.0	100 %
Ghiduk [42]	6376.54	4910.2	2.849	0.005	100 %
Manikumar [47]	5236.8	4507.5	1.133	0.259	100%
Kumar [10]	4921.2	3833.3	0.525	0.600	100%
Mishra [28]	8068.4	8590.8	1.025	0.107	100%
Sahoo [27]	6659.0	5224.0	1.525	0.160	100%
Bao [29]	4083.4	3544.4	0.223	0.823	100%
AGA	3984.0	3230.1	0.0	1.0	100 %

Table 6. Fibonacci program.

Algorithm	Mean	std	ttest	pvalue	Percentage path coverage
Suresh [39]	4228.7	1811.4	5.619	0.009	100 %
Shimin [40]	1772.3	1021.0	3.132	0.003	100 %
Ghiduk [42]	1420.1	743.1	2.413	0.029	100 %
Manikumar [47]	2111.9	1160.2	4.381	0.005	100%
Kumar [10]	1188.3	334.2	0.835	0.405	100%
Mishra [28]	4392.62	1689.7	6.342	0.0	100%
Sahoo [27]	1826.4	1174.9	3.415	0.004	100%
Bao [29]	1275.26	358.6	1.211	0.328	100%
AGA	933.68	301.9	0.0	1.0	100 %

References

[1] McMinn, Phil. "*Search-based software test data generation: a survey.*" Software testing, Verification and reliability 14, no. 2 (2004): 105-156.

[2] Lonetti, Francesca, and Eda Marchetti. "*Emerging software testing technologies.*" In Advances in computers, vol. 108, pp. 91-143. Elsevier, 2018.

[3] Civicioglu, Pinar, and Erkan Besdok. "*A conceptual comparison of the Cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms.*" Artificial intelligence review 39, no. 4 (2013): 315-346.

[4] McGinley, Brian, John Maher, Colm O'Riordan, and Fearghal Morgan. "*Maintaining healthy population diversity using adaptive crossover, mutation, and selection.*" IEEE Transactions on Evolutionary Computation 15, no. 5 (2011): 692-714.

[5] Kim, Su Yong, Sungdeok Cha, and Doo-Hwan Bae. "*Automatic and lightweight grammar generation for fuzz testing.*" Computers & Security 36 (2013): 1-11.

[6] Khan, Rijwan, Mohd Amjad, and Akhilesh Kumar Srivastava. "*Optimization of automatic generated test cases for path testing using genetic algorithm.*" In 2016 Second International Conference on Computational Intelligence & Communication Technology (CICT), pp. 32-36. IEEE, 2016.

[7] Damia, Amir Hossein, and Mohammad Mehdi Esnaashari. "*Automated Test Data Generation Using a Combination of Firefly Algorithm and Asexual Reproduction Optimization Algorithm.*" International Journal of Web Research 3, no. 1 (2020): 19-28.

[8] Pachauri, Ankur, and Gaurav Mishra. "*A path and branch based approach to fitness computation for program test data generation using genetic algorithm.*" In 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), pp. 49-55. IEEE, 2015.

[9] Jiang, Shujuan, Jiaojiao Shi, Yanmei Zhang, and Han Han. "*Automatic test data generation based on reduced adaptive particle swarm optimization algorithm.*" Neurocomputing 158 (2015): 109-116.

[10] Kumar, Sumit, Dilip Kumar Yadav, and Danish Ali Khan. "*A novel approach to automate test data generation for data flow testing based on hybrid adaptive PSO-GA algorithm.*" International Journal of Advanced Intelligence Paradigms 9, no. 2-3 (2017): 278-312.

[11] Khan, Rijwan, Mohd Amjad, and Akhlesh Kumar Srivastava. "*Optimization of automatic test case generation with cuckoo search and genetic algorithm approaches.*" In Advances in Computer and Computational Sciences, pp. 413-423. Springer, Singapore, 2018.

[12] Gupta, Meenakshi, and Garima Gupta. "*Effective test data generation using genetic algorithms.*" Journal of Engineering, Computers & Applied Sciences 1, no. 2 (2012): 17-21.

[13] Sharifipour, Hossein, Mojtaba Shakeri, and Hassan Haghighi. "*Structural test data generation using a memetic ant colony optimization based on evolution strategies.*" Swarm and Evolutionary Computation 40 (2018): 76-91.

[14] Rao, K. Koteswara, G. S. V. P. Raju, and Srinivasan Nagaraj. "*Optimizing the software testing efficiency by using a genetic algorithm: a design methodology.*" ACM SIGSOFT Software Engineering Notes 38, no. 3 (2013): 1-5.

[15] Singh, Bindhyachal Kumar. "*Automatic efficient test data generation based on genetic algorithm for path testing.*" International Journal of Research in Engineering & Applied Sciences 2, no. 2 (2012): 1460-1472.

[16 Liu, Dan, Xuejun Wang, and Jianmin Wang. "*Automatic Test Case Generation based on Genetic Algorithm.*" Journal of Theoretical & Applied Information Technology 48, no. 1 (2013).

[17] Latiu, Gentiana Ioana, Octavian Augustin Cret, and Lucia Vacariu. "*Automatic test data generation for software path testing using evolutionary algorithms.*" In 2012 Third International Conference on Emerging Intelligent Data and Web Technologies, pp. 1-8. IEEE, 2012.

[18] Varshney, Sapna, and Monica Mehrotra. "*Automated software test data generation for data flow dependencies using genetic algorithm.*" International Journal of Advanced Research in Computer Science and Software Engineering 4, no. 2 (2014): 472-479.

[19] Zhu, Xiao-mei, and Xian-feng Yang. "*Software test data generation automatically based on improved adaptive particle swarm optimizer.*" In 2010 International Conference on Computational and Information Sciences, pp. 1300-1303. IEEE, 2010.

[20] Noei, Shirin, Mohammadreza Parvizimosaed, and Mohammadreza Noei. "*Longitudinal Control for Connected and Automated Vehicles in Contested Environments.*" Electronics 10, no. 16 (2021): 1994.

[21] Aleti, Aldeida, and Lars Grunske. "*Test data generation with a Kalman filter-based adaptive genetic algorithm.*" Journal of Systems and Software 103 (2015): 343-352.

[22] Yang, Shunkun, Tianlong Man, Jiaqi Xu, Fuping Zeng, and Ke Li. "*RGA: A lightweight and effective regeneration genetic algorithm for coverage-oriented software test data generation.*" Information and Software Technology 76 (2016): 19-30.

[23] Yang, Shunkun, Tianlong Man, Jiaqi Xu, Fuping Zeng, and Ke Li. "*RGA: A lightweight and effective regeneration genetic algorithm for coverage-oriented software test data generation.*" Information and Software Technology 76 (2016): 19-30.

[24] Pachauri, Ankur, and Gursaran Srivastava. "*Automated test data generation for branch testing using genetic algorithm: An improved approach using branch ordering, memory and elitism.*" Journal of Systems and Software 86, no. 5 (2013): 1191-1208.

[25] Mann, Mukesh, Pradeep Tomar, and Om Prakash Sangwan. "*Test Data Generation Using Optimization Algorithm: An Empirical Evaluation.*" In Soft Computing: Theories and Applications, pp. 679-686. Springer, Singapore, 2018.

[26] Myers, Glenford J., Corey Sandler, and Tom Badgett. The art of software testing. John Wiley & Sons, 2011.

[27] Sahoo, Rashmi Rekha, and Mitrabinda Ray. "*PSO based test case generation for critical path using improved combined fitness function.*" Journal of King Saud University-Computer and Information Sciences 32, no. 4 (2020): 479-490.

[28] Mishra, Deepti Bala, Rajashree Mishra, Kedar Nath Das, and Arup Abhinna Acharya. "*Test case generation and optimization for critical path testing using genetic algorithm.*" In Soft computing for problem solving, pp. 67-80. Springer, Singapore, 2019.

[29] Bao, Xiaoan, Zijian Xiong, Na Zhang, Junyan Qian, Biao Wu, and Wei Zhang. "*Path-oriented test cases generation based adaptive genetic algorithm.*" PloS one 12, no. 11 (2017): e0187471.

[30] Surendran, Anupama, and Philip Samuel. "*Evolution or revolution: the critical need in genetic algorithm based testing.*" Artificial Intelligence Review 48, no. 3 (2017): 349-395.

[31] Grefenstette, J., 2000. Rank-based selection. Evolutionary computation, 1, pp.187-194.

[32] Razali, Noraini Mohd, and John Geraghty. "*Genetic algorithm performance with different selection strategies in solving TSP.*" In Proceedings of the world congress on engineering, vol. 2, no. 1, pp. 1-6. Hong Kong: International Association of Engineers, 2011.

[33] Bullnheimer, Bernd, Richard F. Hartl, and Christine Strauss. "*A new rank based version of the Ant System. A computational study.*" (1997).

[34] Liang, Haibo, Jialing Zou, Kai Zuo, and Muhammad Junaid Khan. "*An improved genetic algorithm optimization fuzzy controller applied to the wellhead back pressure control system.*" Mechanical Systems and Signal Processing 142 (2020): 106708.

[35] Noei, Mohammadreza, and Mohammad Saniee Abadeh. "*A genetic asexual reproduction optimization algorithm for imputing missing values.*" In 2019 9th International Conference on Computer and Knowledge Engineering (ICCKE), pp. 214-218. IEEE, 2019.

[36] Kim, Chiho, Rohit Batra, Lihua Chen, Huan Tran, and Rampi Ramprasad. "*Polymer design using genetic algorithm and machine learning.*" Computational Materials Science 186 (2021): 110067.

[37] Wei, Han, Hua Bao, and Xiulin Ruan. "*Genetic algorithm-driven discovery of unexpected thermal conductivity enhancement by disorder.*" Nano Energy 71 (2020): 104619.

[38] Hamdia, Khader M., Xiaoying Zhuang, and Timon Rabczuk. "*An efficient optimization approach for designing machine learning models based on genetic algorithm.*" Neural Computing and Applications 33, no. 6 (2021): 1923-1933.

[39] Suresh, Yeresime, and Santanu Ku Rath. "*A genetic algorithm based approach for test data generation in basis path testing.*" arXiv preprint arXiv:1401.5165 (2014).

[41] Reddy, G. Thippa, M. Praveen Kumar Reddy, Kuruva Lakshmanna, Dharmendra Singh Rajput, Rajesh Kaluri, and Gautam Srivastava. "*Hybrid genetic algorithm and a fuzzy logic classifier for heart disease diagnosis.*" Evolutionary Intelligence 13, no. 2 (2020): 185-196.

[42] Ghiduk, Ahmed S. "*Automatic generation of basis test paths using variable length genetic algorithm.*" Information Processing Letters 114, no. 6 (2014): 304-316.

[43] Newman, Michael. "*Software errors cost us economy $59.5 billion annually.*" NIST Assesses Technical Needs of Industry to Improve Software-Testing (2002).

[44] Ammann, Paul, and Jeff Offutt. Introduction to software testing. Cambridge University Press, 2016.

[45] Xiao, Man, Mohamed El-Attar, Marek Reformat, and James Miller. "*Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques.*" Empirical Software Engineering 12, no. 2 (2007): 183-239.

[46] Hinterding, Robert, Zbigniew Michalewicz, and Agoston E. Eiben. "*Adaptation in evolutionary computation: A survey.*" In Proceedings of 1997 Ieee International Conference on Evolutionary Computation (Icec'97), pp. 65-69. IEEE, 1997.

[47] Manikumar, T., A. John Sanjeev Kumar, and R. Maruthamuthu. "*Automated test data generation for branch testing using incremental genetic algorithm.*" Sādhanā 41, no. 9 (2016): 959-976.

[48] Kumar, Sumit, Dilip Kumar Yadav, and Danish Ali Khan. "*A novel approach to automate test data generation for data flow testing based on hybrid adaptive PSO-GA algorithm.*" International Journal of Advanced Intelligence Paradigms 9, no. 2-3 (2017): 278-312.

[49] Mansouri, Ardeshir, Mohammadreza Noei, and Mohammad Saniee Abadeh. "*Predicting Hospital Length of Stay of Neonates Admitted to the NICU Using Data Mining Techniques.*" In 2020 10th International Conference on Computer and Knowledge Engineering (ICCKE), pp. 629-635. IEEE, 2020.

[50] Damia, Amirhosein, Mehdi Esnaashari, and Mohammadreza Parvizimosaed. "*Adaptive Genetic Algorithm Based on Mutation and Crossover and Selection Probabilities.*" In 2021 7th International Conference on Web Research (ICWR), pp. 86-90. IEEE, 2021.

[51] Damia, Amirhosein, Mehdi Esnaashari, and Mohammadreza Parvizimosaed. "*Automatic Web-Based Software Structural Testing Using an Adaptive Particle Swarm Optimization Algorithm for Test Data Generation.*" In 2021 7th International Conference on Web Research (ICWR), pp. 282-286. IEEE, 2021.

[52] Parvizimosaed, Mohammadreza, Mohammadreza Noei, Mohammadmostafa Yalpanian, and Javad Bahrami. "*A Containerized Integrated Fast IoT Platform for Low Energy Power Management.*" In 2021 7th International Conference on Web Research (ICWR), pp. 318-322. IEEE, 2021.

[53] Esnaashari, Mehdi, and Amir Hossein Damia. "*Automation of Software Test Data Generation Using Genetic Algorithm and Reinforcement Learning.*" Expert Systems with Applications (2021): 115446.

[54] McCabe, Thomas J. "*A complexity measure.*" IEEE Transactions on software Engineering 4 (1976): 308-320.

[55] Saadtjoo, M. A., and S. M. Babamir. "*Optimizing Cost Function in Imperialist Competitive Algorithm for Path Coverage Problem in Software Testing.*" Journal of AI and Data Mining 6, no. 2 (2018): 375-385.

آزمون نرم‌افزار با استفاده از الگوریتم ژنتیک تطبیقی

امیرحسین دمیا*، مهدی اثنی عشری و محمدرضا پرویزی مساعد

دانشکده مهندسی کامپیوتر، دانشگاه خواجه نصیر الدین طوسی، تهران، ایران.

ارسال ۲۰۲۰/۰۹/۰۱؛ بازنگری ۲۰۲۱/۰۵/۰۷؛ پذیرش ۲۰۲۱/۰۷/۱۴

چکیده:

در آزمون نرم افزار ساختاری، تولید داده های آزمون ضروری است. مساله تولید داده های آزمون یک مساله جستجو است و برای حل این مساله می‌توان از الگوریتم‌های جستجو استفاده کرد. الگوریتم ژنتیک یکی از پرکاربردترین الگوریتم‌ها در این زمینه است. تنظیم پارامترهای الگوریتم ژنتیک به افزایش اثربخشی این الگوریتم کمک می‌کند. در این مقاله، الگوریتم ژنتیک تطبیقی به منظور حفظ تنوع جمعیت در تولید داده های آزمون بر اساس معیار پوشش مسیر استفاده می‌شود، که نرخ بازترکیبی و جهش را با شباهت بین کروموزوم‌ها و مقدار برازندگی کروموزوم محاسبه می‌کند. آزمایشات انجام شده نشان می‌دهد که این روش برای تولید داده‌های آزمون سریعتر از سایر نسخه‌های الگوریتم ژنتیک ارائه شده توسط دیگران است.

کلمات کلیدی: آزمون نرم افزار، تولید داده آموزش، پوشش مسیر، الگوریتم جست و جو، الگوریتم ژنتیک.