2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC) | 979-8-3503-2697-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/COMPSAC57700.2023.00245

A Test Case Prioritization Based on Genetic Algorithm With Ant Colony and Reinforcement Learning Improvement

Yu Yang, Lu Wang, Na Cha, Hua Li
Inner Mongolia University
Inner Mongolia,China
csyangyu@mail.imu.edu.cn,{cswanglu,cslihua}@imu.edu.cn, cn19940123@163.com

***Abstract*—In order to improve the efficiency of regression testing in the cloud-network convergence platform, a test case prioritization method based on reinforcement learning and a genetic algorithm is proposed. The classical genetic algorithm of initial population and selection operations are improved by incorporating an ant colony algorithm of solutions to form a part of the starting population in the genetic algorithm. The selection process employs an "elite retention strategy" to avoid the classical genetic algorithm of the problem of getting trapped in locally optimal solutions. The improved algorithm is applied to test the cloud-network convergence platform, and the optimization-seeking abilities of the classical genetic algorithm, the ant colony genetic algorithm, and the reinforcement learning-based ant colony genetic algorithm are compared and analyzed. The findings reveal that the reinforcement learning-based ant colony genetic algorithm outperforms the other two algorithms by finding the best test case for the test case prioritization problem.**

Keywords—test case prioritization, reinforcement learning, genetic algorithm, ant colony algorithm

I. INTRODUCTION

With the combination of Software Defined Network (SDN) and cloud computing technologies, it has a better way to pool computing, storage, and networking resource in a cloud environment and increase the utilization of various resources in a data center. However, the regression testing related to cloud network integration becomes more complicated because of the deployment of increasingly complex software systems. At this stage, selecting and prioritizing test cases from a large collection of cases is crucial to locate faults in software rapidly[1]. Although there are various optimization-seeking Artificial Intelligence (AI) algorithms available, we still need to find an efficient and simple method to implement with fewer requirements for historical data and high learning capabilities.

The paper is organized as the following section: Section II introduces the background. Section III proposes the problem and the solution which includes a basic algorithm and improvement direction. Section IV describes our experiments and analysis. Finally, the conclusion is presented in Section V.

II. BACKGROUND

A. Genetic algorithm

Genetic algorithm is a heuristic search algorithm that builds on the laws of biological evolution in the natural world [2]. The core of genetic algorithm is the process of evaluation, selection, crossover, and variation of individual fitness. However, the presence of super chromosomes at the beginning of the genetic algorithm will make the diversity of the population decrease thus causing the algorithm to converge prematurely and fall into a state of local optimum solution. Moreover, the merit of the initial population of the genetic algorithm also determines the performance of the genetic algorithm, and the diversity and distribution of the initial population largely affect the convergence speed of the genetic algorithm[3].

B. Ant colony algorithm

The ant colony algorithm is widely used in finding an optimum after it is proposed by the Italian scholar Marco Dorigo in 1992 [4]. It is also an optimization algorithm based on population strategies and is easy to combine with other algorithms with high robustness. The ant colony algorithm is also a heuristic algorithm like the genetic algorithm.

C. Reinforcement learning

Reinforcement learning[5] is a special type of machine learning method used to describe and solve the problem of maximizing the reward or achieving a specific goal through learning strategies of an intelligent agent (Agent) during its interaction with the environment[6].

The basic model for reinforcement learning is Markov Decision Processes [7](MDPs), and there are two main types of reinforcement learning methods: model-based dynamic planning methods and model-free reinforcement learning methods[8],Dynamic planning methods can be grouped into strategic iterative methods and value iterative methods. Model-free reinforcement learning approach can be grouped into Monte Carlo methods and time-difference approaches.

Among them, the Q-Learning is a modeless reinforcement learning algorithm with MDP as the theoretical basis [9]. Its action function value formula is as formula (1)

979-8-3503-2697-0/23/$31.00 ©2023 IEEE
DOI 10.1109/COMPSAC57700.2023.00245

$$Q(s,a) = R(s,a) + \gamma V^* \quad (1)$$

In Equation (1), $Q(s,a)$ represents the value of the action function for performing action a in state s is equal to the discounted value of the reward value $R(s,a)$ and the optimal policy V^*, γ denotes the discount factor. The optimal strategy at this point can be rewritten as Equation (2):

$$\pi^*(s) = arg\max_a Q(s,a) \quad (2)$$

Equation (2) indicates that the optimal strategy depends on the performance of the function of action a in state s. The optimal strategy is the one that makes $Q(s,a)$ to be the maximum value.

In the Q-Learning algorithm, a Q-Table is defined to store the action function values $Q(s,a)$ of the intelligence, and the action function values can be expressed as an iterative form of the Q-Table update shown in Equation (3):

$$Q^*(s_t,a_t) = R(s_t,a_t) + \gamma Q^*(s_{t+1},a_{t+1}) \quad (3)$$

In Equation (3), $Q^*(s_t,a_t)$ denote the optimal action function value at the t moment, $R(s_t,a_t)$ is the reward value at the t moment, $Q^*(s_{t+1},a_{t+1})$ denotes the optimal action function value at the $t+1$ moment, and γ denotes the discount factor for the most optimal strategy. At each step of the iteration, Equation (3) can be specified in the following form of Equation (4):

$$Q^*(s_t,a_t) \leftarrow R(s_t,a_t) + \gamma \max_a Q^*(s_{t+1},a) \quad (4)$$

The intelligence continuously updates the Q-Table according to Equation (4), which indicates that the value of the element located at a position (s_t,a_t) in the Q-Table is related to the performance of the reward value $R(s_t,a_t)$ and the value of the maximum action function at the next state after discounting, and γ is the discount factor.

The proposed environment under test in this paper is a platform with OpenStack and OpenDaylight cloud-network convergence, and it is necessary to give a collection of test cases firstly and then make a priority execution selection.

Since the cloud-network convergence platform built relevant operations on resources through the Horizon page, relevant test cases can be generated based on the UML activity diagram of the Horizon page. The process is shown in Fig.1.

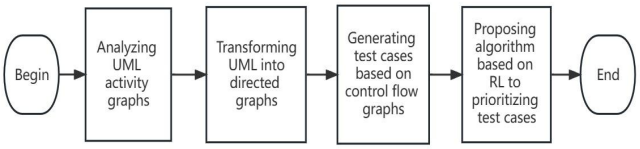

Fig. 1. Test case prioritization process

Since there are many more mature methods for test generation[9], we assume that we have a test case set here. Because any path of a directed graph from the starting node to the ending node can constitute a test case in which each type of node carries a different weight, all that needs to be done in the following part are how to design the rules for assigning weights and how to prioritize the test cases.

III. Problems and Solutions

When conducting related research based on OpenStack and OpenDaylight platform, we need to perform multiple regression tests on its basic functions. In order to find the fault occurrence point faster, we have to select the testing cases for priority execution.

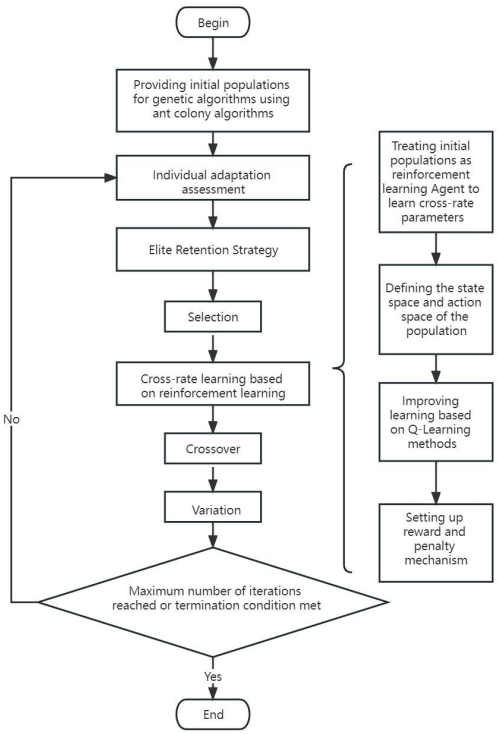

Fig. 2. Basic flow chart of Ant Colony-Genetic Algorithm based on Reinforcement Learning

A. Problems

The purpose of test case prioritization is to pick the test cases to be executed from the set of target test cases, so the method can be viewed as a single-objective optimizing problem at the beginning, the classical genetic algorithm is chosen as the basis for problem-solving, and the test case preference process is also evolved based on genetic operations. Since the genetic algorithm has the disadvantage of easily losing good solutions and falling into local optimum, the way of generating the offspring population needs to be improved, the ant colony algorithm is used to provide a partially better solution to the original population of the genetic algorithm. To avoid the loss of high-quality solutions and a roulette wheel approach to accelerate the elimination of junk genes during the selection and crossover process of genetic operations, we use an elite retention strategy. Finally, in order to make the crossover rate of the population no longer depend on manual setting, the crossover rate in the genetic algorithm is adaptively learned based on reinforcement learning techniques. The algorithm flow of this work is illustrated in Figure 2.

In Figure 2, it can be seen that the ant colony algorithm is introduced at the beginning of the genetic algorithm, and the result of the ant colony algorithm is used as the initial solution of the genetic algorithm, on which the genetic operation can avoid some risks caused by the inappropriate choice of initial values. At the end of the execution of the ant colony algorithm, part of the solution is used as part of the initial population of the genetic algorithm, and the rest can be randomly generated by the random function as a way to ensure the diversity of species.

B. Construction of the fitness function

To obtain prioritization execution order of test paths an algorithm is proposed based on reinforcement learning and ant colony genetic. The core of the genetic algorithm is the process of evaluation, selection, crossover and variation of individual fitness. Firstly, the magnitude of the fitness values of the populations are compared and the current optimal solution is updated; Then, the optimal solution is judged to see whether it belongs to the target test path set, and if it does, the path is removed from the target test path set, otherwise the execution continues iteratively. To construct the fitness function, there are three steps.

1) Calculation of Weight

After converting the UML activity diagram into the corresponding directed graph, weights need to be assigned to each node in the directed graph. The assignment of weights is based on the importance of the nodes. In the activity diagram, the branch node generally determines the direction of the activity and is the most critical node, therefore it should be the largest weight percentage. The rules for assigning weights are shown in Table I.

TABLE I. NODE WEIGHT ALLOCATION TABLE

Activity Graph Nodes	*Weights*
Initial Node	0
Active Nodes	2
Branch Nodes	5
Fork/Confluence Nodes	3
Merge Nodes	4
Ending Nodes	0

According to the rules for assigning weights in Table I, the weights of each chromosome in the population can be derived, and the formula is shown in (5).

$$f(x) = \sum_{i=1}^{n} w_i \ , \forall c \in P \tag{5}$$

In Equation (5),$f(x)$ denotes the weight of the chromosome c , w_i denotes the weight of the i[th] node, P represents the population, c represents each chromosome belonging to P, and the formula indicates that the weight of each chromosome is derived by accumulating the weights of the n nodes that make up the chromosome.

2) Design of constraint functions

The purpose of designing the constraint function is to add penalty terms to the objective function and optimize the target function. Firstly, the entire control flow graph is traversed with DFS to derive the set of all non-repeating paths, which is set as the target test path set S ,s_p denotes a path in the target test path set, and the total number of nodes on the path is denoted by $|S_p|$[12]. Also, assume L is a traversed path.

The formulation of constraints is related to the similarity between the target test path and the traversed path, the target test path $s_p$is compared with the currently traversed path$L$, and the number of identical nodes is recorded between the first node of the two paths and the first different node that appears, which is denoted as γ_p , then the similarity of S_p and L can be expressed as shown in Equation (6).

$$path\_sim = \frac{\gamma_p}{|S_p|}, \ p = 1,2,\ldots,n \tag{6}$$

From Equation (6), it can be seen that when the value of γ_p is larger, the number of paths S_p and L with the same nodes is higher, and the path similarity is higher. When $\gamma_p / |S_p| = 1$, it means that the traversal path $L$is exactly the same as the target test path S_p . Therefore, the value range of this $path\_sim$ is [0,1].

Using the above constraint $path\_sim$ as the penalty function $\psi(x)$, the penalty function formula is shown in (7).

$$\psi(x) = \bigvee_{p=1}^{n} \frac{\gamma_p}{|S_p|} - 1 \tag{7}$$

As the target test path contains multiple paths $s_1, s_2, s_3, \ldots, s_n$, the path traversed here must be one of the target test paths. This can be seen from the formula(6), if the path traversed is one of the target test paths, then $\gamma_p / |S_p| = 1$, which means $\psi(x) = 0$, the penalty function will not affect the value of the objective function; otherwise $\psi(x)$ is less than 0, in this case, the penalty function will play its role in reducing the fitness value of the objective function, and as a test criterion.

3) Building the fitness functions

The main objective here is to optimize the order of test case execution by making the last obtained test case as the first one to be executed, which is still essentially an objective optimization problem. Therefore, the problem can be expressed as a single-objective optimization problem with a penalization function, finding the test path with the largest weight from the set of target test paths, and the penalty function ensures that the traversed paths must belong to one of the target test paths. To mirror the tradeoff relationship between the weight function $f(x)$ and the penalty function $\psi(x)$, coefficients α and β are added respectively. The fitness function $F(x)$ is shown in Equation (8).

$$F(x) = \alpha f(x) + \beta \psi(x), \alpha > 0, \beta > 0 \tag{8}$$

In Equation (8), α and β are greater than 0, and need to be set in advance.

The operation of the genetic algorithm is as usual and will not be repeated here.

C. Cross-rate self-learning strategy based on reinforcement learning

In prior experiments, most of the parameters of the genetic algorithm are based on research experience. With the in-depth

study of genetic algorithms, the setting of its parameters has gradually tended to be an adaptive strategy. And choosing the appropriate crossing rate and variability can improve the efficiency of the algorithm to a great extent. Therefore, this work improves the crossing rate of the genetic algorithm based on the Q-Learning algorithm of reinforcement learning, aiming at using reinforcement learning to realize the self-learning of the population crossover rate. The flow of the self-learning strategy of cross-rate parameters based on reinforcement learning is shown in Figure 3.

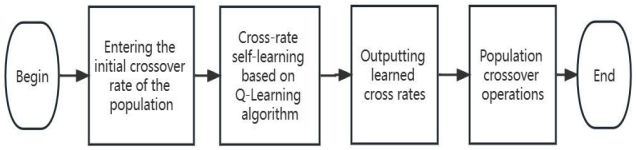

Fig. 3. Flow chart of self-learning strategy of cross rate parameter based on reinforcement learning

The main idea of the population cross-rate self-learning strategy based on reinforcement learning is as follows: the population in the genetic algorithm is regarded as an intelligence body in reinforcement learning, and the learning of cross-rate parameters is regarded as the goal of the intelligence body, and the state space and the action space of the population is defined. The Q-Learning method of reinforcement learning is used for continuous improvement and learning, and the reward and punishment mechanism is set in the learning process, so that the intelligence body learns and explores in the expected direction, and obtains better cross-rate parameters.

The cross-rate self-learning strategy based on reinforcement learning can not only realize the independent learning of cross-rate but also update the cross-rate parameters dynamically. The reinforcement learning steps are as follows:

1) Setting state space and action space:

- State space: The state space is composed of cross-rate parameters, and in this work, the state space consists of two decimal places between [0.75,0.99].
- Action space: The action space consists of three numbers, 1, -1, and 0. Where -1 means punishment, 0 means no change and 1 means reward. The action space is changed by 0.01 each time, when the action space indicates reward, the next state will increase by 0.01, and when it indicates punishment, the next state will decrease by 0.01. Therefore, the relationship between this state and the next state is as shown in Equation (9).

$$S_{t+1} = S_t + action \times 0.01 \tag{9}$$

2) Setting reward and penalty mechanism:

The reward and penalty mechanism is the key element when learning cross-rate through Q-Learning. The reward and penalty mechanism in this work is mainly related to the following indicators:

- The target value of the optimal chromosome in the current population;
- The average target value of the current population;
- The genotype of the population;

The standard variance of the target values of individuals in the population.

To avoid the algorithm from slipping into local convergence, the reward and punishment mechanism is changed with 50 steps as the dividing line. The specific conditions for setting the reward and penalty mechanism and the process is:

- When the number of generations of the current population is less than 50, the reward and penalty mechanism is related to the target function of the best individual in the population and the group's average target value.
- When the number of generations of the current population is greater than 50, the reward and penalty mechanism is not only related to the target function of the best individual in the population and the group average target value but also related to the standard variance of the target function value of the population and the genotype of the population.

The evaluation indicator of the population is extremely important when a new crossover rate is generated. The evaluation index is set to {0, 1, 2}, with {0} indicating no change for the better, {1} indicating change for the better, and {2} indicating no requirement. For the test case prioritization problem, the higher the target function value, the higher the weight of the path, and it means that the path should be tested first. Therefore, the larger the target function value and the average target function value, the better the quality and stability of the population when the standard variance of the population and the genotype of the population are reduced.

Following the above evaluation indexes, the reward and penalty mechanism is shown in Table II.The four evaluation indexes are expressed in four digits, the first two denote the optimal target function value and the average target function value of the population, and the last two denote the standard variance of the population and the genotype of the population.

3) Explore and utilize:

The action selection strategy of the Q-Learning algorithm is the $\varepsilon-$ greedy strategy, and the evaluation strategy is the greedy strategy, which may fall into the local optimum if it is too greedy in the training process. Therefore, in order to balance the exploration and the utilization process, the greedy strategy is used to prevent the premature convergence of the Q-table, and the ε parameter is used to control the greediness of the intelligence so that the intelligence has a certain probability of randomly selecting actions in addition to passing the Q-table to complete further exploration of the environment.

TABLE II. REWARD AND PENALTY MECHANISM SETTING TABLE

Portfolio of evaluation indicators	*Number of iterations less than 50*	*Number of iterations more than 50*
1122	+15	+12
1022	+5	+5

Portfolio of evaluation indicators	*Number of iterations less than 50*	*Number of iterations more than 50*
0122	+3	+5
1111	-10	+30
2210	-10	-5
2201	-10	-5
2211	-10	-5
Others	-10	-25

4) The overall process of cross-rate parameter self-learning strategy:

Reinforcement learning is an environment-based continuous trial-and-error and improvement learning process, in which the initial state needs to be repeated several times for experiments. After learning, the average or the maximum of multiple crossover rates corresponding to the better Q value is taken and allowed to be the parameter for the next input of the genetic algorithm.

In addition to that some parameters of the algorithm need to be initialized, such as the values of the discount rate, learning rate, exploration rate, need to be set in advance, and the action space, state space, and initial Q-Table need to be initialized.

IV. Experiment

A. Experimental design

To assess the effectiveness of the reinforcement learning and genetic algorithm-based test case prioritization method proposed in this study, the UML activity diagrams for creating instances and creating mirror functions in the cloud-network convergence platform were validated individually.

The genetic algorithm (GA), ant colony genetic algorithm (AC_GA), and reinforcement learning-based ant colony genetic algorithm (RL_GA) are compared experimentally for creating instances and creating UML activity diagrams for mirroring function of cloud-network convergence platform, respectively. The population size is set as 50, the maximum number of iterations is 9999, the crossover rate of the genetic algorithm is 0.8, the mutation rate is 0.05, the discount rate γ is 0.9, the learning rate α is 0.05 and the greed coefficient ε is 0.8. Firstly, the UML activity diagram is converted into the corresponding control flow diagram, and secondly, the DFS algorithm is used to generate the set of target test paths based on the control flow diagram. To prove the capability of the algorithm, the genetic algorithm, the ant colony genetic algorithm, and the reinforcement learning-based ant colony genetic algorithm were run ten times for the create instance activity and create mirror activity, respectively, while recording the number of iterations and the average number of iterations when the optimal target test path was found. The experimental results are shown in Tables III. Suppose C express the number of iterations of creating instance activities and M express the number of iterations of mirroring activities for three algorithms.

TABLE III. Comparison of the number of iterations of creating instance activities and mirroring activities for three algorithms

Number of experiments	*Genetic algorithm*		*Ant colony genetic algorithm*		*Reinforcement learning-based ant colony genetic algorithm*	
	C	M	C	M	C	M
1	117	89	23	113	45	25
2	233	179	101	124	29	19
3	137	221	89	75	14	67
4	79	241	54	131	67	89
5	87	178	78	158	29	78
6	165	87	126	57	37	21
7	197	100	148	89	51	53
8	141	127	98	91	71	33
9	178	141	159	39	99	103
10	69	129	45	67	83	43
Average	140.3	149.2	92.1	94.4	52.5	53.1

From Table III, it can be seen that the average number of iterations of the genetic algorithm is more than 140, and the average number of iterations of the ant colony genetic algorithm is in the interval [90,100], while the average number of iterations of the reinforcement learning-based ant colony genetic algorithm is in the interval [50,60]. From the average number of iterations to find the target test path is significantly lower than the other two algorithms, so it has better optimal search performance.

In Figure 4 and Figure 5, the distribution of the 10 experiments on the graph shows that the number of iterations for the genetic algorithm is in the interval [60,250], the number of iterations for the ant colony genetic algorithm is in the interval [20,160], and the number of iterations for the reinforcement learning based ant colony genetic algorithm is in the interval [0,110]. From Figure 4 and Figure 5, the reinforcement learning-based ant colony genetic algorithm exhibits a lower average number of iterations than the genetic algorithm and ant colony genetic algorithm, making it possible to identify the target test path in fewer iterations.

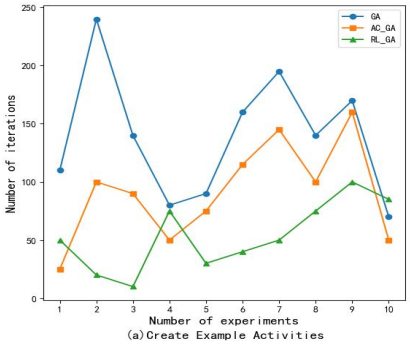

Fig. 4. Iteration comparison of creating instance activity for three algorithms

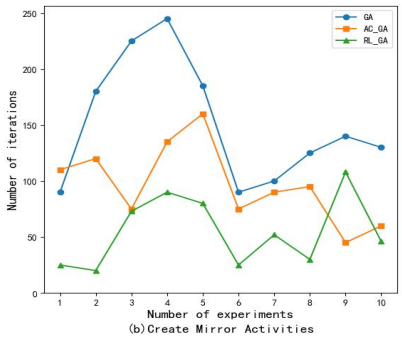

Fig. 5. Iteration comparison of creating mirror activity for three algorithms

Based on the two functions of creating mirrors and creating instances, the success rates of the three algorithms that can find the target test path within a limited number of iterations of 50, 100, and 200 are counted and compared. The success rates of finding the target test paths are shown in Tables IV. It can be seen that when the number of iterations does not exceed 50, the success rates of the reinforcement learning-based ant colony genetic algorithm in finding the target test paths are higher than the other two algorithms.

TABLE IV. COMPARISON OF THE SUCCESS RATE OF FINDING THE TARGET TEST PATH FOR CREATING INSTANCES AND MIRRORING ACTIVITIES

algorithm	***50 times***		***100 times***		***200 times***	
GA	0%	0%	20%	30%	90%	80%
AC_GA	20%	10%	60%	70%	100%	100%
RL_GA	50%	50%	100%	100%	100%	100%

B. Analysis of experimental results

The classical genetic algorithm is easily trapped in the local optimum in the process of iteration. After improving its initial population, it avoids falling into local optimum to partly speed up the convergence rate. On this basis, the settings of the cross-rate parameters of the genetic algorithm are optimized, so that the cross-rate can be learned autonomously based on reinforcement learning, and the cross-rate parameters that are suitable for the population can be found better. The above comparison shows that the ant colony genetic algorithm based on reinforcement learning has better finding ability and can find the optimal solution in a shorter number of iterations for the test case preference problem based on the UML activity diagram.

V. CONCLUSION

To enhance the effectiveness of regression testing of the cloud-network convergence platform, we investigate the test case prioritization technique in conjunction with the cloud-network environment. In terms of test case priority execution, this work improves the genetic algorithm by introducing an ant colony algorithm in the algorithm, which is used to generate the initial solution of the genetic algorithm and prevent the genetic algorithm from falling into the local optimum. The evolution of the genetic algorithm is based on three genetic operators of selection, crossover, and variation, and the selected values of genetic operators largely affect the performance of the genetic algorithm, we propose an adaptive strategy of crossover rate parameters based on Q-Learning algorithm, which makes the crossover probability parameters is no longer a fixed value based on experience, but a crossover probability suitable for that generation of the population is generated by reinforcement learning, so as to achieve the purpose of optimizing the parameters. In addition, an "elite retention strategy" is used in the selection process in order to preserve the quality of solutions. For the purpose of verifying the performance of the algorithm, the reinforcement learning-based ant colony genetic algorithm is applied to the test environment. The experimental results show that the presented algorithm has good results in finding the best solution to the test case preference problem.

ACKNOWLEDGMENT

We do appreciate the great support of the National Natural Science Foundation of China (No.62262047, 62066034), the Inner Mongolia Science & Technology Plan (No.201802028,2020GG0186), the Inner Mongolia discipline inspection and supervision big data laboratory open project fund (No.IMDBD2020011).

REFERENCES

[1] Mukherjee R, Patnaik K S. A survey on different approaches for software test case prioritization[J]. Journal of King Saud University-Computer and Information Sciences, 2021,33(9):1041-1054.

[2] Deng L, Chen H, Liu H, et al. Overview of UAV path planning algorithms[C]//2021 IEEE International Conference on Electronic Technology, Communication and Information (ICETCI). IEEE, 2021:520-523.

[3] Lingling Yin, Jianfeng Su. Exploring the convergence of genetic algorithms based on initial populations[J]. Journal of Taiyuan Normal College (Natural Science Edition),2020,19(01):54-57.

[4] JunYang ,Wenkang Kong ,Qiuye Sun . A review on the application of intelligent algorithms for fault recovery in distribution networks containing distributed power sources[J]. Control and Decision, 2019, 34(9):1809-1818.

[5] XinWang ,FangWang. A review of research on dynamic pricing strategies based on reinforcement learning[J]. Computer Applications and Software, 2019, 36(12):1-6.

[6] KanghaoWang ,HaibingYin ,XiaofengHuang . A policy gradient-based target tracking method[J]. Journal of Zhejiang University (Engineering Edition), 2020, 54(10):1923-1928.

[7] Puterman M L. Markov decision processes: discrete stochastic dynamic programming[M]. John Wiley & Sons, 2014.

[8] XianGuo, YongchunFang. In-depth reinforcement learning: an introduction to principles [M]. Beijing: Electronic Industry Press,2018

[9] Li-MingWang. Research on material scheduling method of zero inventory management in manufacturing enterprises[J]. Value Engineering, 2019,38(23):126-129

[10] Wei C Y, Jahromi M J, Luo H, et al. Model-free reinforcement learning in infinite-horizon average-reward markov decision processes[C]//International conference on machine learning. PMLR, 2020:10170-10180.

[11] Del Carpio P M. Towards more Interactive Recordings for E-leaming Using Browser Automation[C]//2021 IEEE Engineering International Research Conference (EIRCON).IEEE, 2021:1-4。

[12] Gong D, Zhang Y. Generating test data for both path coverage and fault detection using genetic algorithms[J]. Frontiers of Computer Science, 2013, 7(6):822-837.