Automated Test Case Generation Based on Differential Evolution With Relationship Matrix for iFogSim Toolkit

Han Huang, *Member, IEEE*, Fangqing Liu, *Student Member, IEEE*, Zhongming Yang, and Zhifeng Hao, *Member, IEEE*

***Abstract*—Fog computing plays an important role in industrial and information process. The programs in fog computing, such as iFogSim toolkit, usually contain some infeasible paths (paths that cannot be covered), which makes it impossible to compare algorithm in models that require covering all paths. In this paper, we proposed a mathematical model to build automated test case generation based on path coverage (ATCG-PC) in fog computing programs as a single-objective problem. Single objective helps to reduce the cost of evaluation functions, which is proportional to the number of test cases. When infeasible paths are contained in tested programs, algorithms can also be compared in this model. In this paper, classical differential evolution (DE) is used to solve the ATCG-PC. However, it is difficult for DE to use generated test cases covering remaining paths in the ATCG-PC of fog computing. Therefore, we proposed a test-case-path relationship matrix to empower DE (RP-DE). Experiment results show that RP-DE uses significantly less test cases and achieves higher path coverage rate than compared state-of-the-art algorithms.**

***Index Terms*—Automated test case generation (ATCG), differential evolution (DE), fog computing, iFogSim, path coverage (PC), relationship matrix.**

Manuscript received May 27, 2018; accepted July 7, 2018. Date of publication July 18, 2018; date of current version November 1, 2018. This work was supported by NSFC-Guangdong Joint Found under Grant U1501254, National Natural Science Foundation of China under Grant 61772225, Guangdong Natural Science Funds for Distinguished Young Scholar under Grant 2014A030306050, the Ministry of Education—China Mobile Research Funds under Grant MCM20160206, Guangdong High-level personnel of special support program under Grant 2014TQ01X664, Guangzhou Science and Technology Program key projects under Grant 201804010276, and International Cooperation Project of Guangzhou under Grant 201807010047. Paper no. TII-18-1331 *(Corresponding author: Fangqing Liu.)*

H. Huang is with the School of Software Engineering, South China University of Technology, GuangZhou 510006, China, with the Department of Computer Science, City University of Hong Kong, Hong Kong 999077, China, and also with the Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, College of Computer Science and Technology, Jilin University, Changchun 130012, China (e-mail: hhan@scut.edu.cn).

F. Liu is with the School of Software Engineering, South China University of Technology, GuangZhou 510006, China (e-mail: 564376030@qq.com).

Z. Yang is with the College of Computer Engineering Technical, Guangdong Institute of Science and Technology, Zhuhai 510640, China (e-mail: yzm8008@126.com).

Z. Hao is with the School of Computer, Guangdong University of Technology, GuangZhou, and the School of Mathematics and Big Date, Foshan University, Foshan 528000, China (e-mail: zfhao@fosu.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2018.2856881

I. Introduction

FOG computing is a framework of edge computing and mobile computing, while the software or application designed based on fog computing systems contains some problems. For example, the overall testing cost is too expensive. Testing is an important process in the development cycle of fog computing systems. It is also a labor-intensive work which costs almost 50% of the entire development cost [1], [2].

Many researchers focus on using functional testing to evaluate their designed architectures or frameworks in fog computing. For example, Tang *et al.* [3] used test data to evaluate the recognition performance of their proposed architecture in a simulated smart pipeline monitoring system. Hu *et al.* [4] built test databases by collecting face images of tested individuals and evaluated the performance of their scheme. Zhu *et al.* [5] proposed two collaborative location-based sleep scheduling schemes to integrate wireless sensor networks and mobile cloud computing. The performance of the scheduling schemes was evaluated based on simulation requests of mobile users. Gupta *et al.* [6] proposed a fog computing simulator which aimed at modeling internet of things (IoT) [7] and fog environments. This simulator could detect the resource management in terms of latency, energy consumption, and cost. These research works mentioned above focus on evaluating the functional performance of fog architectures or frameworks. However, structural testing is also a major challenge in fog computing. It reveals logical defects of software.

Automated test case generation (ATCG) is one of the most demanding aspects of structural testing activities [1], [2]. ATCG belongs to unit testing [8], [9], which means that ATCG needs to analyze tested functions. Given a system under test (SUT) and its coverage criterion, the coverage situation is finite and discrete. Therefore, ATCG cannot be derivable. Most of the ATCG problems work on some coverage criteria, such as branch, statement, or path coverage. Among those kinds of coverage criteria, path coverage has been considered as the most critical and challenging one [10], [11]. The SUT in fog computing systems also contains some problems in ATCG based on path

coverage (ATCG-PC). For example: 1) some existing infeasible paths; 2) the cost of evaluating test case increases with the number of path exploding.

Several kinds of mathematical models have been built with the fitness function. Fraser *et al.* [11] proposed a model whose objective is to minimize the number of elements in test suit while it satisfies coverage criterion at the same time. Yao *et al.* [12] provided a model for the problem of ATCG-PC. In this model, each target path will have its fitness function. Only the test case that covers the corresponding path could have the optimal fitness value. Both of the above mentioned models use fitness function to evaluate test cases. However, more fitness functions are needed with the number of paths exploding in ATCG-PC. The cost for evaluating the performance of compared algorithms will increase with the number of paths increasing at the same time. Mala *et al.* [13] built the mathematical model for the multiobjective optimization problem, which aimed at maximizing the number of covered paths and minimizing the number of test cases at the same time. Huang *et al.* [14] built a useful mathematical model for ATCG-PC. The target of this model is to minimize the number of test cases for covering all paths. However, the performance of the compared algorithms cannot be evaluated, if no algorithm covers all paths or the SUT contains some infeasible paths.

Static analysis and search-based algorithms are usually used to solve ATCG-PC. Static analysis concentrates on analyzing the code of tested function and generating test cases automatically [2]. Symbolic execution is one kind of classical static analysis method; it uses symbols instead of concrete values to represent the values of tested functions [15]. However, two problems exist when applying this technique to real world programs. First, it is difficult to apply the static analysis in fog computing function with complex constraints. Second, the computational capability for using static analysis to solve ATCG-PC in fog computing is expensive.

Search-based algorithms are proposed to solve ATCG-PC and overcome the shortcomings that exist in static analysis. Harman *et al.* [16] proposed that researchers tended to use evolutionary algorithms to solve ATCG among different search-based algorithms. Wang *et al.* [17] and Suresh *et al.* [18] used genetic algorithm (GA) as the solution method for ATCG-PC. Zhang *et al.* [19] introduced a type of multipopulation GA, while Bouchachia *et al.* [20] proposed an immune strategy to improve GA with global search capacity. The algorithms mentioned above focus on improving the searching ability of algorithms for solving ATCG-PC. Some researchers supposed that suitable fitness function could help to find remaining paths in ATCG-PC. Lin *et al.* [21] proposed a GA using Hamming distance to evaluate test cases for ATCG-PC. Mala *et al.* [13], [22] indicated that artificial bee colony (ABC) could outperform ant colony optimization. Huang *et al.* [14] proposed a differential evolution (DE) based on self-adaptive fitness function for ATCG-PC. The techniques mentioned above –[13], [14], [21], [22] emphasize on designing suitable fitness functions combined with state-of-the-art algorithms. However, the relationship between test case and path is also essential for testing fog computing systems. If some particular test case dimensions are changed, the offspring individuals may cover different paths. If we can build a data-driven model and apply it to solve ATCG-PC, many test cases could be saved.

In this paper, we propose a mathematical model for ATCG-PC in fog computing systems. In addition, a DE empowered by a test-case-path relationship matrix programming differential evolution (RP-DE) is introduced. The proposed model just needs to evaluate each test case once. The proposed objective criterion (path coverage rate) in the model makes comparison possible even when infeasible paths exist in SUT. RP-DE has a test-case-path relationship matrix to guide the direction of iteration while many extra test cases are saved.

The rest of this paper is organized as follows. Section II introduces the background of fog computing, DE, and ATCG problem. Section III proposes a mathematical model of ATCG-PC in fog computing. Section IV describes a DE empowered by a test-case-path relationship matrix. Section V details our experiment results, and shows the efficiency of our proposed RP-DE algorithm. Some influence factors of RP-DE are also introduced in this section. Section VI concludes this paper.

II. Background

In this section, we introduce fog computing, ATCG problem, and the DE algorithm.

A. Fog Computing and Related Toolkit

Fog computing [23], [24] is a kind of distributed framework, which focuses on providing localization services, and enabling low latency and context awareness. Cloud aims at providing global centralization services.

Fog computing paradigms are needed for some applications that require emergency response with low latency. Low communication delay between the cloud and application is also required. Gupta *et al.* [6] proposed a simulator toolkit, called iFogSim, to model the fog environment and measure the impact of resource management techniques in fog computing systems. iFogSim simulates different scenarios such as the resource management and the scheduling between edge and cloud resources. There are several important structures in iFogSim, such as IoT sensors, IoT actuators, IoT data stream, fog device, and so on.

In [6], iFogSim shows the sequence of tuple emission and its subsequent execution. Several functions, such as "send," "transmit," and "executeTuple" are implemented for transferring information or other processes. Those functions are classical and common in fog computing systems. We assume that those functions reflect the software structure in most fog computing systems. As a result, we select six most common functions as benchmark functions in this paper.

B. DE

The DE [25]–[28] algorithm, as a metaheuristic algorithm, uses linear combination calculation with local individuals to generate offspring. In each iteration of DE, every individual will generate a candidate solution with other local solutions

selected randomly in the same population [29], [30]. DE also uses similar operations, such as mutation and crossover [26], [31], to generate offspring individuals when compared with other evolutionary algorithms.

The formula of DE mutation is shown as

$$V_i = X_i + F \times (X_j - X_k) \tag{1}$$

where V_i is the candidate solution, X_i is the mutated solution, X_j and X_k are solutions selected randomly, while F is a factor parameter in range of [0,1]. The formula of DE crossover is shown as

$$x_{ij} = \begin{cases} v_{ij}, & \text{rand}_j^i \le P_c \text{ or } j = j_{\text{rand}} \\ x_{ij}, & \text{otherwise} \end{cases} \tag{2}$$

where v_{ij} is the value of the jth dimension in the ith candidate solution, rand_j^i is a random number in [0,1], j_{rand} represents a random dimension number, P_c indicates the crossover probability, while x_{ij} is the original jth dimension value of mutated solution.

C. Problem Description of ATCG

Before we focus on ATCG-PC, some definitions regarding ATCG-PC, such as 1) test case; 2) function; 3) path; and 4) path coverage, will be introduced.

Definition 1 (test case): A test case is usually a vector $X_i = (x_{i1}, x_{i2}, ..., x_{in})$ with n elements, where each dimension is an integer or float number in the range of its domain.

Definition 2 (function): A function is a string of code, which has an input test case and its single output result, in a fog computing system.

Definition 3 (path): A path is a sequence of running direction of vertices, starting at some initial vertices and ending at some final vertices. Each vertex will have its branch condition in the tested fog computing function.

Definition 4 (path coverage): Path p_j is covered by a test case X_i, which means that X_i generates the same running direction with path p_j in the tested function.

The objective of ATCG-PC is to use the minimum test cases to cover the most paths, while branch coverage focuses on covering most branches. Both of those coverage criteria need to cover more target paths or branches with the minimum test cases. However, if the test cases $\bar{X} = \{X_1, \dots X_L\}$ could cover all paths, $\bar{X}$ has at least one proper subset which covers all branches. While each test case can cover single path, a test case usually covers many branches. This evidence shows that more test cases are needed for solving ATCG-PC.

Several challenges exist in ATCG-PC. First, the relationship between test case and its covered path is unknown. Second, the method which generates offspring test cases and covers remaining paths with the used test cases needs to be determined. In order to solve the two challenges, two missions should be completed. First, a suitable mathematical model of ATCG-PC should be proposed to reveal the relationship between test case and path in ATCG-PC. Second, a reasonable algorithm needs to be designed to solve ATCG-PC based on the proposed mathematical model.

III. Mathematical Model of ATCG-PC in Fog Computing

In this section, we will introduce the mathematical model of ATCG-PC in fog computing system. Four important parts of this mathematical model are shown in the following paragraphs: 1) path encoding; 2) basic variables; 3) objective; and 4) constraints.

Path encoding shows the way to represent paths in ATCG-PC. Notice that there may be several vertices for a given fog computing function. All those vertices need to be allocated with a unique number from 1 to k, where k is the number of vertices in the tested function. We find that each test case covers exactly one path, and a path p is consisted by a sequence of vertices. As a result, all paths in tested fog computing functions can be represented by a sequence string with k characters. The empty character of path p in the jth character means that p skips the jth vertex, while nonempty characters indicate different running directions of path p in the jth vertex.

Given a tested fog computing function, $\bar{X} = \{X_1, X_2, ..., X_N\}$ represents the set of candidate test cases in the decision space. Notice that the number of candidate test cases N is a large number and the cost for exploring all test cases is expensive. $P_0 = \{p_1, p_2, ..., p_L\}$ indicates the set of paths in a tested fog computing function, where L is the number of paths in a tested fog computing function. The objective of ATCG-PC in fog computing functions can be reformulated as follows:

Maximize c

Subject to

$$c = \frac{l}{L} \times 100\% \tag{3}$$

$$l = \sum_{j=1}^{L} \min\{1, \sum_{i=1}^{T} \Theta_{ij}\} \tag{4}$$

$$T \le \text{Max} \tag{5}$$

$$\Theta_{ij} = \begin{cases} 1, & \text{test case } X_i \text{ covers path } p_j \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

$$\sum_{j=1}^{L} \Theta_{ij} = 1 \tag{7}$$

$$X_i = (x_{i,1}, ..., x_{i,k}, x_{i,k+1}, ..., x_{i,n})$$

$$x_{i,\mu} \in Z, \quad lb_\mu \le x_{i,\mu} \le ub_\mu$$

$$x_{i,\mu'} \in R, \quad lb_{\mu'} \le x_{i,\mu'} \le ub_{\mu'} \tag{8}$$

$$1 < i \le N; \mu = 1, 2, ..., k; \mu' = k+1, k+2, ..., n. \tag{9}$$

Constraints (3) and (4) define the calculation of objective value c, where c represents the proportion of covered paths among all paths in a fog computing SUT, l is the number of paths covered by the generated test cases, and L is the number of paths in a tested fog computing function. Constraint (5) ensures that the number of generated test cases T will not exceed Max.

Max represents the maximum acceptable test cases for solving ATCG-PC. The value of Max is usually significantly less than N. The value of T is equal to the times of evaluating test cases which is a fair criterion to show the computation time of ATCG-PC. Max is usually a constant according to the related work [11], [12], [32]. Constraints (6) and (7) require each test case that could cover exactly one path. Constraints (8) and (9) define the decision space of the decision variables. Some variables of X_i are integers, while others are real numbers. The objective of our proposed mathematical model is to achieve maximum value of c within the maximum test case consumption.

Our proposed mathematical model is used to solve two problems in former models [11]–[14]. First, the mathematical models in [11] and [12] model ATCG-PC as a multitasking problem that each path needs to be covered as single task. A new generated test case should be evaluated by the fitness functions of all paths. We model ATCG-PC as a single-objective problem and each test case will only be evaluated once. The cost of evaluation is saved. Second, [13] and [14] both require 100% path coverage with minimum test cases. However, if all algorithms under comparison cannot achieve 100% coverage or some infeasible paths are contained in the tested program, the performance of algorithms being compared cannot be evaluated. By contrast, the performance of algorithms can be evaluated in our proposed model with path coverage rate when tested programs contain infeasible paths.

Constraint (4) shows that if many test cases cover the same path, only one of them will be recorded. However, this phenomenon causes too many extra test cases. Our proposed algorithm should make use of the used test cases and extract data-driven knowledge to cover remaining paths.

IV. RP-DE Algorithm for ATCG in Fog Computing

In this section, a DE algorithm empowered by the test-case-path relationship matrix (RP-DE) will be introduced. We suppose that the test-case-path relationship matrix could analyze the used test cases and provide the optimization direction for DE. This matrix helps DE to achieve higher values of c within maximum test cases in tested fog computing functions.

A. Test Case and Path Relationship Matrix

Constraints (6) and (7) show that each test case covers only one path. However, many test cases may cover the same path so that it generates significant extra consumption. Obviously, reducing the number of test cases covering paths found already is the key for reducing the consumption of ATCG-PC in fog computing. As a result, we design a test-case-path relationship matrix R to empower DE for ATCG-PC.

We will discuss some detailed information about the test-case-path relationship matrix R. R contains n lines and m columns, where n is the number of dimensions in each test case, and m is the number of vertices in tested fog computing functions. R will be set to be a zero matrix at the initial operation. The value in the ith line and the jth column of matrix R represents the correlation coefficient between the ith test case dimension and the jth vertex in tested fog computing function. Those coefficients are generated based on the generated test cases.

Algorithm 1: Searching based on R matrix.

Input : R matrix, the optimized test case x_{old}, target path p_{target}

1 **for** *each* $j \in \{1, 2, ..., m\}$ **do**
2 **if** $p^j_{x_{old}} \neq p^j_{target}$ **then**
3 **for** i=*1 to* n **do**
4 $weight_i = R_{i,j}$
5 **end**
6 Use roulette-wheel selection to get a candidate dimension Dim with $weight_i$ where $i = 1, ..., n$.
7 $step = (ub_{Dim} - lb_{Dim})/4$.
8 **while** $p^j_{x_{best}} \neq p^j_{target}$ *and* $step > 0$ **do**
9 Use $step$ as the searching length to search dimension Dim based on quartile.
10 Record five offspring test cases from x_1 to x_5, where they are upper bound, upper quartile, median, lower quartile and lower bound respectively.
11 **for** i=*1 to 5* **do**
12 Update R matrix based on **Algorithm 2**.
13 **end**
14 Record the test case with highest fitness value with x_{best} among $x_1, x_2, ..., x_5$.
15 $step = step/4$.
16 **end**
17 **end**
18 **end**

Output: a test case x_{target} covering p_{target}

19 * ub and lb are the upper bound and lower bound of domain in test cases.

The pseudocode of searching based on the R matrix is shown by Algorithm 1. We suppose that the optimized test case is X_{old}, $p_{X_{\text{old}}}$ is the path encoding covered by X_{old}, and $p^j_{X_{\text{old}}}$ represents the jth character in the path encoding of $p_{X_{\text{old}}}$. As shown in Line 2 of Algorithm 1, each vertex between the path encodings of pXold and $p$target will be in comparison. If the jth vertex is different, the data of jth column in R matrix will be collected and roulette wheel selection is used to generate a candidate dimension Dim. The Dimth dimension of X_{old} will be explored based on quartile [33] iteratively, as shown in Line 7 to 15. Quartile is a useful statistical method which effectively reflects the distribution of data. This process has very high probability to generate offspring test cases which has the same path encoding with p_{target} in jth vertex. After iterating all vertices of the path covered by X_{old} and p_{target}, RP-DE can generate X_{target} which covers p_{target}.

The efficiency of Algorithm 1 is ensured by the correlation coefficient collected in former populations. As a result, the maintenance of matrix R is one key concept of RP-DE. Each time we generate a new test case X_i, the operation of updating R would be triggered

$$R^t_{i,j} = \begin{cases} R^{t-1}_{i,j} + 1, & p^j_{X_{\text{old}}} \neq p^j_{X_i} \\ R^{t-1}_{i,j}, & \text{otherwise} \end{cases} \tag{10}$$

where i and j indicate the row index and column index in matrix R, respectively, and t is the iteration time of the optimization algorithm. Row index i is equal to the selected optimized dimension Dim.

Algorithm2: Updating R matrix.

Input : R matrix, the optimized test case x_{old}, the offspring test case x_i

1 **for** *each* $j \in \{1, 2, ..., m\}$ **do**
2 **if** $p^j_{x_{old}} \neq p^j_{x_i}$ **then**
3 Use Equation (10) to update R.
4 **end**
5 **end**

Output: The updated R matrix

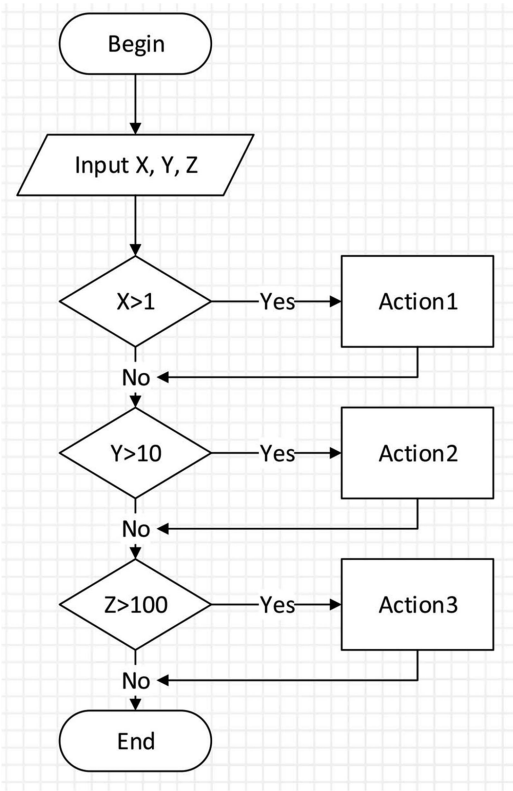


Fig. 1. Flow control graph of a tested function. The process from Action1 to Action3 represents some operations with X, Y, and Z.

Algorithm 2 shows the process of updating the R matrix. When an offspring test case is generated by Algorithm 1, Algorithm 2 would be triggered to evaluate whether to update the R matrix. This process is shown in Line 12 of Algorithm 1. The updating process is based on the different vertexes with the path encoding between X_{old} and its offspring test cases. Only when X_{old} generates offspring test case with different path, the relationship matrix R will be updated. The updating operation of R matrix is essential because it can collect data and generate useful knowledge from populations of former generations. It also empowers DE in tested fog computing functions.

B. Example of R Matrix

An example will show the searching and updating processes in Algorithms 1 and 2, respectively. Given a SUT in Fig. 1, all input variables are integers in the range of [1,200]. If we use "0" and "1" to represent the direction of "Yes" and "N o," respectively, all paths shown in Fig. 1 can be represented by a three-bit binary string.

TABLE I
EXAMPLE FOR R MATRIX FOR TESTED FUNCTION IN FIG. 1

101	0	0
0	45	0
0	0	146

* The value in the ith line and the jth column of matrix R represents the correlation coefficient between the ith test case dimension and the jth vertex.

Assume a condition that all paths have been covered except for "011", and the relationship matrix R is shown in Table I. The value of c is equal to 87.5%. In this study, the test case $X_{\text{old}} = (50,3,64)$, which covers "111", should be optimized to cover "011". First, the path encoding of "111" and "011" will be compared. The running direction of first vertex is different. As shown in lines 3–6 of Algorithm 1, a candidate dimension Dim will be selected based on roulette wheel selection. Dim is clearly equal to one, as the weight vector is (101,0,0). Then, the Dimth dimension of X_{old} will be explored based on quartile iteratively. The process is shown as Line 8 to 16 of Algorithm 1. After several rounds of iteration, an offspring test case $X_{\text{target}} = (1,3,64)$ will be found. The value of c will achieve 100%. At the same time, Algorithm 2 will be triggered to update R matrix. The value of "101" will be updated to "102."

C. Framework of RP-DE Algorithm

With the proposed test-case-path relationship matrix, many evolutionary algorithms, such as GA, PSO [28], and ABC can be combined to solve ATCG-PC. We need to find one kind of evolutionary algorithm to use the proposed test-case-path relationship matrix. The performance of matrix R is based on collecting correlation coefficient between test case dimensions and paths. We chose DE as the optimized algorithm.

We assume that DE can make the most use of the proposed matrix. The flowchart of our proposed RP-DE algorithm is shown in Fig. 2. There are three important operations in RP-DE with the tested fog computing function:

1) population initialization;
2) DE mutation and crossover;
3) optimization with R matrix.

In the initial population of DE algorithm, the population will be randomly initialized in the range of their domain. The formula of the initial DE population is shown as

$$x_{i,j} = rand() * (ub_j - lb_j) + lb_j \tag{11}$$

where $rand()$ is a stochastic value in [0,1], ub_j and lb_j are the upper bound and lower bound of the jth dimension of test cases, and $x_{i,j}$ represents the jth dimension value of the ith test case. The population will be evaluated after initialization, while the paths covered by those test cases will also be recorded. This process will update the value of θ_{ij} in Constraint (6).

The iteration of DE mutation, crossover, and optimization with R matrix helps RP-DE to achieve the highest value of c. Two operations are iteratively executed under the iteration of

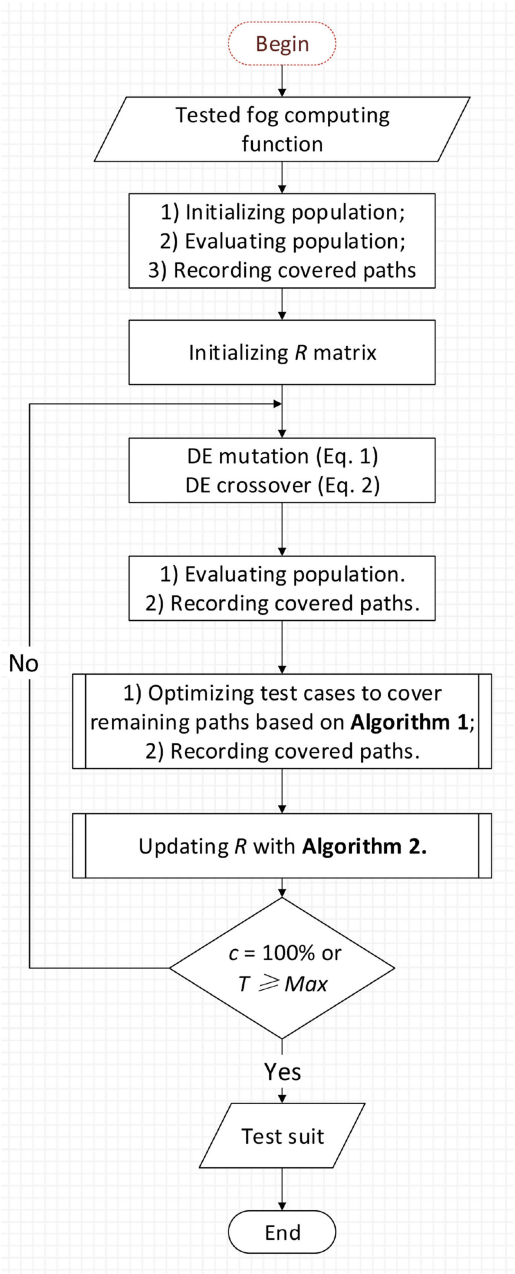


Fig. 2. Flowchart of proposed RP-DE algorithm.

RP-DE algorithm. First, RP-DE algorithm will use (1) and (2) to generate offspring test cases. Second, the offspring population will be optimized with the R test-case-path relationship matrix based on Algorithms 1 and 2. DE mutation and crossover will search the decision space globally, while optimization with R matrix accelerates the convergence speed of RP-DE. A test case, which covers the target path, will be found after several rounds of iterations.

In order to make full use of the R matrix, a suitable target path should be selected for each optimized test case. The optimized test case X_i and its covered path p_{X_i} will be compared with remaining paths based on their path encoding. Every remaining path will be assigned to a weight value based on the number of characters between p_{X_i} and the remaining paths. The more of the same characters they have, the higher weight value will be set to the remaining paths. The target path will be selected with the weight value of remaining paths based on the roulette wheel selection.

Constraint (8) shows that each input variable has single interval. RP-DE can generate offspring test cases within their interval. If the variable x_{ij} is an integer number, x_{ij} will first be updated as a real number in its interval. The final value of x_{ij} will be set to $\lfloor x_{ij} + 0.5 \rfloor$. By contrast, real number variables will be ensured like the operation in classical DE [34].

V. Experiments and Analysis of Results

In this section, the objective of experiment is to support that test-case-path relationship matrix can save many test cases for ATCG-PC. This claim will be proved by discussing the performance of RP-DE in comparison to other published evolutionary algorithms for ATCG-PC.

A. Experimental Setup

As we introduced in Section II, fog computing toolkit iFogSim is a kind of architecture to simulate and model IoT and fog environments. Several functions, such as "send," "transmit," and "executeTuple", achieve the information exchanging and other reaction operations. Those functions are selected as the benchmark functions, while some basic information is shown in Tables II and III.

Table II shows the number of lines, input test case dimensions, and paths of benchmark functions. The functional descriptions of those functions are also introduced. Table III represents the size of the decision space in each benchmark function and the probability of each path to be covered by a randomly generated test case. The probabilities of paths, which are smaller than 1.00E-10, are in bold font. If the number of paths with very small probability (e.g., less than 1.00E-10) increases, it is difficult for algorithms to have a high value of c.

There are four groups of experiments in this section.

1) The parameter sensitive analysis about factor parameter F and crossover probability P_c in RP-DE will be discussed.
2) RP-DE algorithm will be compared with DE [14] based on the mathematical model in [14] and our proposed model.
3) RP-DE will be also compared with Immune GA (IGA) [20], ABC [35], particle swarm optimization (PSO) [36], their variants with test-case-path relationship matrix (RP-IGA, RP-ABC, RP-PSO) and a type of stochastic strategy (Random) based on our proposed mathematical model.
4) RP-DE will be compared with competitive swarm optimizer (CSO) [37] and PSO with convergence speed controller (PSO-CSC) [38].

The first experiment will analyze how F and P_c influence the performance of RP-DE. The second experiment is to show how the test-case-path relationship matrix helps DE to save extra

TABLE II
BENCHMARK FUNCTIONS IN IFOGSIM

Program	Loc	Dim	Description	Path
transmit	30	3	transmit sensor data to another sensor	2
send	47	2	send packaged tuple from sensor to fog-device or from one fog-device to another fog-device	9
processEvent	67	7	fog-device process events	9
executeTuple	41	7	use tuple processing logic to update device power consumption	5
checkCloudletCompletion	43	5	be called on fog-device on completion of execution of the tuple	6
getResultantTuple	73	8	return the processed tuple to applications	7

*Loc, Dim and Path represent the number of lines, input dimensions and paths in each tested function.
*Description introduces the task of tested function in iFogSim toolkit.

TABLE III
DECISION SPACE AND PROBABILITY OF PATHS IN BENCHMARK FUNTIONS

Program	ID	size	probability of each path
transmit	1	1.68E+07	5.96E-08; 1.00E+00
send	2	4.00E+12	0; 0; 0; 2.50E-07; 2.50E-07; 2.50E-07; 2.50E-01; 2.50E-01; 5.00E-01;
processEvent	3	6.00E+19	1.25E-06; 1.25E-05; 1.25E-05; 6.25E-02; 6.25E-02; 1.25E-01; 1.25E-01; 1.25E-01; 5.00E-01
executeTuple	4	8.44E+14	**5.33E-15;** 4.47E-08; 4.47E-08; 2.50E-01; 7.50E-01
checkCloudletCompletion	5	6.71E+07	4.47E-08; 4.47E-08; 2.50E-01; 2.50E-01; 2.50E-01; 2.50E-01
getResultantTuple	6	1.69E+15	**1.78E-15; 3.55E-15;** 9.93E-09; 1.99E-08; 2.98E-08; 8.94E-08; 1.00E+00

* "probability of each path" indicates the probability of each path to be covered by a randomly generated test case based on uniform distribution in tested function. The result sorts from small to large.

TABLE IV
EXPERIMENT PARAMETER SETTING

Parameter Name	Value	Parameter Name	Value
F (RP-DE. DE)	0.5	P_c (RP-DE. DE)	0.2
crossover probability (IGA)	0.8	mutation probability (IGA)	0.1
limit (ABC)	2	bee number (ABC)	50
c1;c2 (PSO, PSO-CSC)	1.5;2	weight value (PSO, PSO-CSC)	0.4
phi (CSO)	0.8	period (PSO-CSC)	150
threshold of rule1 (PSO-CSC)	0.9	threshold of rule2 (PSO-CSC)	0.001
population size	50		

test cases and higher value of c. The objective of the third is to show the efficiency of RP-DE algorithm in ATCG-PC of fog computing systems. All compared algorithms are designed for solving ATCG-PC. It also shows that DE is necessary for test-case-path relationship matrix. In the final experiment, RP-DE will be compared with CSO [37] and PSO-CSC [38]. The benchmark functions are selected from iFogSim [6] and some open-source programs [39], [40].

We deployed the algorithms under comparison on the same computational environment. The experimental computer was equipped with Intel i5-4590 3.30 GHz, 8 GB memory, and Windows Education 10 OS. The number of population is 50 [14]. The number of maximum test cases in our proposed model and in [14] is 3.00E+05 [32] and 1.00E+08 [14], respectively. The number of trials in four groups of experiments is 30. All the other parameters are arranged with the same values which are set in [14], [20], [35]–[38]. The detailed information is shown in Table IV. The method of significance testing is based on Wilcoxon rank-sum testing [41], [42] with $\alpha = 0.05$.

B. Parameter Sensitive Analysis of RP-DE

Three parameters may influence the performance of DE and RP-DE. Specifically, they are population size, factor parameter F, and crossover probability P_c. To make a fair comparison, all compared algorithms use the same population size in all experiments. However, the factor parameter F and crossover probability P_c are arranged according to the problem. In this subsection, we will analyze the the influence of above two parameters in RP-DE.

In this experiment, F will be set to 0.25, 0.5, and 0.75, while P_c will be set to 0.2 and 0.8, respectively. Table V shows the performance of RP-DE with different values of F and P_c. The performance indicators, which perform best, are noted with bold font. From the value of "+/=/-" in Table V, we can see that there is no significant difference between RP-DE3 and other variants. We conclude that RP-DE is not sensitive to those two parameters in tested benchmark functions.

Among all combinations of F and P_c, we find that the parameter setting in RP-DE3 ($F = 0.5$, $P_c = 0.2$) uses the smallest test cases although F and P_c are not sensitive to the performance of RP-DE. The parameter setting in RP-DE3 is relatively a better choice. In the following experiment, RP-DE will use the parameter setting in RP-DE3.

C. Comparing RP-DE With DE Based on iFogSim Toolkit

In this subsection, our proposed RP-DE algorithm will be compared with DE [14] based on our proposed mathematical model and the model proposed in [14]. The experiment result is shown in Tables VI and VII.

Test-case-path relationship matrix saves more test cases than DE in tested benchmark functions based on our proposed mathematical model. As shown in Table VI, RP-DE achieves 100% ave.c in four benchmark functions within six tested functions of 30 trials. DE only has the highest ave.c in one tested function "transmit". On average, RP-DE saves 1.93E+03 test cases in "transmit". RP-DE also achieves higher value of ave.c than DE in the remaining two benchmark functions "send" and "getResultantTuple". Moreover, RP-DE uses significantly less test cases than DE in all benchmark functions except for "send". RP-DE also significantly outperforms DE in benchmark function "checkCloudletCompetion". We find that test-case-path relationship matrix helps DE to save a large number of test cases in all tested benchmark functions except for "send".

TABLE V
EXPERIMENT RESULTS OF RP-DE WITH DIFFERENT PARAMETER SETTING

Function	RP-DE1(F=0.25;P_C==0.2)		RP-DE2(F=0.25;P_C==0.8)		RP-DE3(F=0.5;P_C==0.2)		RP-DE4(F=0.5;P_C==0.8)		RP-DE5(F=0.75;P_C==0.2)		RP-DE6(F=0.75;P_C==0.8)	
ID	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)
1	100%	5.39E+03(2.53E+03)	100%	5.11E+03(2.05E+03)	**100%**	**4.23E+03(1.30E+03)**	100%	4.90E+03(1.75E+03)	100%	4.96E+03(1.16E+03)	100%	5.28E+03(1.78E+03)
2	66.7%	3.00E+05(0)	66.7%	3.00E+05(0)	66.7%	3.00E+05(0)	66.7%	3.00E+05(0)	66.7%	3.00E+05(0)	66.7%	3.00E+05(0)
3	100%	1.70E+04(1.11E+04)	100%	1.80E+04(8.98E+03)	**100%**	**1.31E+04(5.05E+03)**	100%	1.43E+04(7.43E+03)	100%	1.65E+04(7.65E+03)	100%	1.41E+04(8.12E+03)
4	94.7%	1.80E+05(9.02E+04)	97.3%	1.20E+05(6.88E+04)	**100%**	**7.10E+04(4.17E+04)**	99.3%	8.81E+04(4.42E+04)	99.3%	7.26E+04(4.95E+04)	100%	7.17E+04(3.62E+04)
5	100%	1.61E+04(7.69E+03)	99.4%	2.39E+04(2.11E+04)	**100%**	**9.17E+03(2.44E+03)**	100%	1.07E+04(2.98E+03)	100%	1.01E+04(3.39E+03)	100%	1.06E+04(4.21E+03)
6	84.9%	2.59E+05(6.00E+04)	89.8%	2.12E+05(8.76E+04)	**98.5%**	**1.59E+05(7.01E+04)**	96.5%	1.63E+05(9.49E+04)	95.6%	1.71E+05(8.15E+04)	96%	1.68E+05(8.37E+04)
+/=/-	0/6/0		0/6/0				0/6/0		0/6/0		0/6/0	

* ave.c: the average value of c in all paths for tested fog computing function;
* ave.T(std.T): the average and standard test case number of each algorithm in tested benchmark functions;
* +/=/-: the number of '+', '=' and '-' represent that RP-DE3 uses significant less, equal and more test cases than other variants of RP-DE with Wilcoxon rand-sum test with $\alpha = 0.05$.

TABLE VI
EXPERIMENT RESULTS OF RP-DE WITH DE BASED ON OUR PROPOSED MODEL

Function	RP-DE		DE	
ID	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)
1	100%	4.20E+03(1.32E+03)	100%	6.13E+03(1.16E+03)+
2	66.7%	3.00E+05(0)	57.6%	3.00E+05(0)=
3	100%	1.41E+04(6.25E+03)	79.9%	3.00E+05(0)+
4	100%	9.40E+04(4.79E+04)	80.7%	2.92E+05(1.55E+04)+
5	100%	9.81E+03(2.94E+03)	98.9%	4.26E+04(4.42E+04)+
6	97.5%	1.57E+05(6.75E+04)	35.6%	3.00E+05(0)+
+/=/-				5/1/0

* ave.c: the average value of c in all paths for tested fog computing function.
* ave.T(std.T): the average and standard test case number of each algorithm in tested benchmark functions.
* +/=/-: the number of "+", "=" and "-" represent that RP-DE uses significant less, equal, and more test cases than compared algorithm with Wilcoxon rand-sum test with $\alpha = 0.05$.

TABLE VII
EXPERIMENT RESULTS OF RP-DE WITH DE BASED ON MODEL IN [14]

Function ID	RP-DE	DE
1	**4.50E+03**	6.38E+03
	(1.63E+03)	(9.44E+02)+
2	/	/
	(/)	(/)=
3	**1.21E+04**	/
	(4.66E+03)	(/)+
4	**7.07E+04**	/
	(3.85E+04)	(/)+
5	**9.98E+03**	3.85E+04
	(2.82+03)	(3.88E+04)+
6	**1.56E+05**	/
	(7.84E+04)	(/)+
+/=/-		5/1/0

* Three lines of data in each tested benchmark function respectively indicate the average and standard test case number in six tested benchmark functions.
* '/' indicates that the compared algorithm does not cover all possible paths within the maximum test cases.
* +/=/-: the number of "+," "=," and "-" represent that RP-DE uses significant less, equal, and more test cases than compared algorithm with Wilcoxon rand-sum test with $\alpha = 0.05$.

As shown in Table III, there are three infeasible paths in this problem. The performance of DE and RP-DE cannot be compared, if the experiment is based on mathematical model in [14]. However, those two algorithms can be compared by the value of c in our proposed model. The highest ave.c value of "send" is 66.7% while the ave.c values of RP-DE and DE are 66.7% and 57.6%, respectively. As a result, RP-DE has the better performance than DE in this problem.

As shown in Table VII, the two lines of data in each tested benchmark function indicate the average and standard test case number in six tested benchmark functions. The performance indicators of algorithms, which perform best in those compared algorithms, are noted with bold font. All paths are possible in those six tested benchmark functions; however, "send" only contains six possible paths. The experiment result shows that RP-DE and DE cannot be compared in tested function with infeasible paths based on model of [14]. Table VII shows that RP-DE uses significantly less test cases than DE in most tested benchmark functions. Our proposed test-case-path relationship matrix R helps the DE algorithm to generate significantly less test cases than DE when the maximum test case number in this model is 10^8.

The experiment results shown in Tables VI and VII indicate that the test-case-path relationship matrix significantly empowers DE in those six tested benchmark functions, no matter based on our proposed model or model in [14]. Test-case-path relationship matrix collects the correlation coefficient between each test case dimension and each path vertex. This matrix will guide the optimization direction for DE. Besides, the proposed model evaluates the path coverage rate (the value of c) and makes comparison possible when infeasible paths exist in tested programs.

D. Comparing RP-DE With Other Evolutionary Algorithms Based on iFogSim Toolkit

In this subsection, RP-DE will be compared with other state-of-the-art algorithms [20], [35], [36] and their variants with relationship matrix. The experiment result is based on our proposed model, which means that the target of each compared algorithms is to cover as many paths as possible.

RP-DE achieves the highest value of c when it is compared with state-of-the-art algorithms in six benchmark functions based on our proposed model. The boxplots of RP-DE and compared state-of-the-art algorithms are shown in Fig. 3. The higher the average value of c means better performance among compared algorithms. Fig. 3 shows that all of the ave.c for RP-DE are 100% in four boxplots (a), (c), (d), and (e). RP-DE contains four trials that not all paths have been covered within maximum test cases in boxplot (f) of Fig. 3. However, RP-DE still covers more paths than all the compared algorithms. There

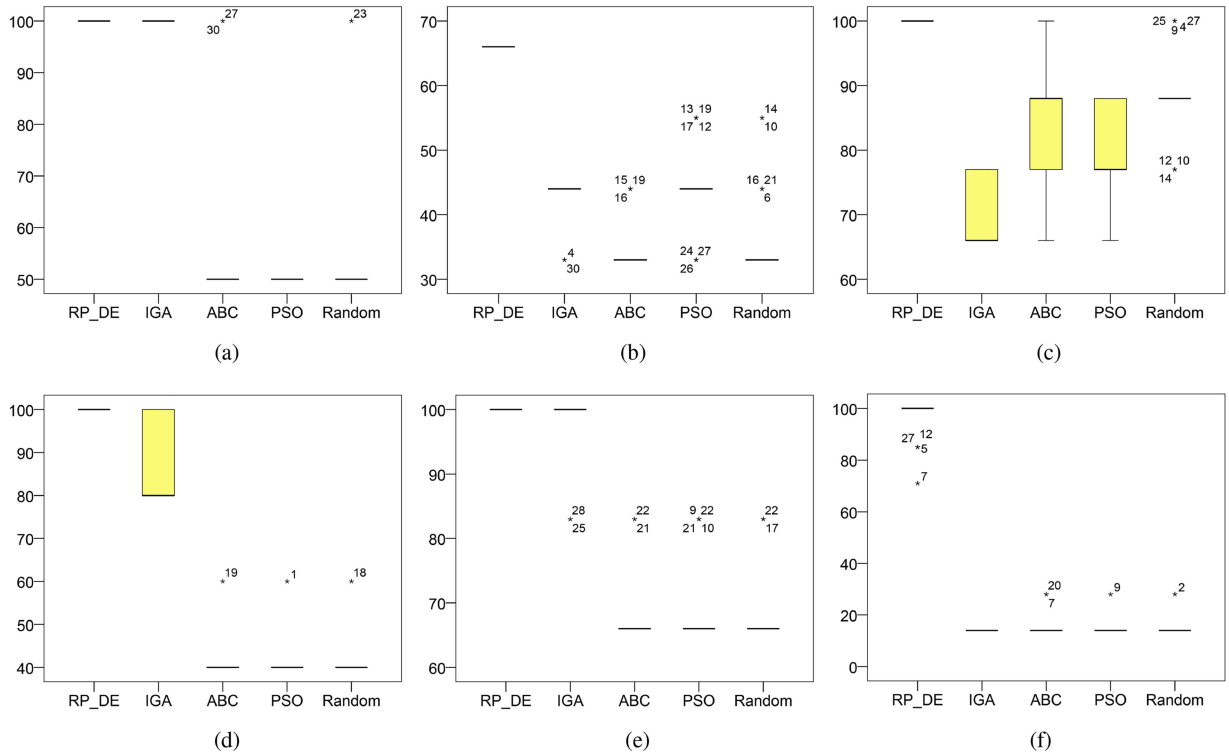


Fig. 3. Plot boxes of c for the algorithms under comparison. (a) trasmitt. (b) send. (c) processEvent. (d) executeTuple. (e) checkCloudletCompletion. (f) getResultantTuple. * The x-axis exhibits compared algorithms; the y-axis indicates the distribution of c.

are only six possible paths among the nine paths in the benchmark function "send". As a result, the highest ave.c is 66.7%; further, RP-DE performs the best in "send" which is shown in Fig. 3(b).

RP-DE uses significantly less test cases than the compared algorithms. The performance of RP-DE shows its efficiency in solving ATCG-PC of tested benchmark functions. Table VIII shows the experiment results of RP-DE under the comparison with other state-of-the-art evolutionary algorithms and their variants based on our proposed model. As shown in Table VIII, RP-DE achieves the highest ave.c in all benchmark functions. Although IGA and RP-IGA also cover all paths in benchmark function "transmit", RP-DE uses significantly less test cases to cover all paths than compared algorithms in this function. RP-DE has the value of ave.c with 66.7% in benchmark function "send", while the highest ave.c in IGA, ABC, and PSO is just 44%. The result of "+/=/-" shows that RP-DE uses significantly less test cases and saves aspect of test cases than compared algorithms in most tested benchmark functions. The results also show that DE is necessary because RP-DE performs best in most cases.

The test-case-path relationship matrix empowers the performance of IGA, ABC, and PSO in solving those benchmark functions. From Table VIII, the variants of IGA, ABC, and PSO achieve higher value of ave. c in most tested functions. Their variants even achieve the highest ave.c in tested function "send" and "processEvent", while the exactl values are 66.7% and 100%, respectively. The relationship matrix makes IGA, ABC, and PSO more competitive in solving ATCG-PC.

E. Comparing RP-DE With PSO-CSC and CSO

In this subsection, RP-DE will be compared with two well-known evolutionary algorithms, CSO [37] and PSO-CSC [38], in iFogSim [6] and some open-source real world programs [39], [40]. From these programs [39], [40], another six functions are selected as the benchmark functions (Table X).

bibclean-2.08 is a program to check syntax and print in BibTeX bibliography files. The two selected benchmark functions are used to check validity of ISBN and ISSN code, respectively. **gimp-2.2.4** is an image manipulation program. Several library functions for color space conversion are chosen. **tiff-3.8.2** is a program for processing images in the Tag Image File Format (TIFF). The function for image placing routines is selected.

The experiment results are shown in Table IX. From the results in Function ID 1–6, RP-DE achieves higher ave.c value and uses significantly less test cases in most functions. In the result of tested function "send", RP-DE has the highest value of ave.c. It means that it covers all feasible paths, while PSO-CSC and CSO only achieve ave.c value with 44.7% and 48.8%, respectively. All of these three algorithms cover all paths in "trasmit." PSO-CSC and CSO use significantly less test cases than RP-DE. It seems that RP-DE has lower performance than PSO-CSC and CSO in some tested functions with small decision space. The experiment results in last six benchmark functions show that the ave.c value of RP-DE is larger or equal to PSO-CSC and CSO. Three benchmark functions in **gimp-2.2.4** both have infeasible paths, while RP-DE still covers the most paths. Especially in tested function "TIFF_GetSourceSamples", RP-DE saves 7.43E+04 test cases on average when it is compared

TABLE VIII
EXPERIMENT RESULTS OF RP-DE WITH OTHER ALGORITHMS BASED ON OUR PROPOSED MODEL

Function	RP-DE		IGA		ABC		PSO	
ID	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)
1	100%	4.20E+03(1.32E+03)	100%	1.00E+04(3.99E+03)+	53.3%	2.96E+05(2.06E+04)+	50%	3.00E+05(0)+
2	66.7%	3.00E+05(0)	43.3%	3.00E+05(0)=	34.8%	3.00E+05(0)=	44%	3.00E+05(0)=
3	100%	1.41E+04(6.25E+03)	70.4%	3.00E+05(0)+	83.3%	2.95E+05(2.21E+05)+	79.2%	3.00E+05(0)+
4	100%	9.40E+04(4.79E+04)	86%	2.30E+05(1.02E+04)+	40.7%	3.00E+05(0)+	40.7%	3.00E+05(0)+
5	100%	9.81E+03(2.94E+03)	98.9%	5.57E+04(5.19E+04)+	67.1%	3.00E+05(0)+	68.3%	3.00E+05(0)+
6	97.5%	1.57E+05(6.75E+04)	14%	3.00E+05(0)+	14.9%	3.00E+05(0)+	14.5%	3.00E+05(0)+
+/=/-				5/1/0		5/1/0		5/1/0

(a)

Function	RP-IGA		RP-ABC		RP-PSO		Random	
ID	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)
1	100%	3.97E+04(1.59E+04)+	56.7%	2.83E+05(3.04E+04)+	58.3%	2.75E+05(4.40E+04)+	51.7%	3.00E+05(0)+
2	66.7%	3.00E+05(0)=	66.7%	3.00E+05(0)=	66.7%	3.00E+05(0)=	35.6%	3.00E+05(0)=
3	100%	1.47E+04(1.43E+04)=	100%	2.44E+03(1.55E+03)-	100%	6.05E+03(5.46E+03)-	88.5%	3.00E+05(0)+
4	58.7%	2.64E+05(6.04E+04)+	44%	3.00E+05(0)+	42.7%	3.00E+05(0)+	40.7%	3.00E+05(0)+
5	96.7%	1.02E+05(7.45E+04)+	75.6%	2.91E+05(1.86E+04)+	76.2%	3.00E+05(0)+	67.1%	3.00E+05(0)+
6	43.9%	2.77E+05(3.84E+04)+	16.8%	3.00E+05(0)+	15.9%	3.00E+05(0)+	14.5%	3.00E+05(0)+
+/=/-		4/2/0		4/1/1		4/1/1		5/1/0

(b)

* ave.c: the average value of c in all paths for tested fog computing function.
* ave.T(std.T): the average and standard test case number of each algorithm in tested benchmark functions.
* +/=/-: the number of "+,""=," and "-" represent that RP-DE uses significant less, equal, and more test cases than compared algorithm with Wilcoxon rand-sum test with $\alpha = 0.05$.

TABLE IX
EXPERIMENT RESULTS OF RP-DE WITH PSO-CSC AND CSO BASED ON OUR PROPOSED MODEL

Function	RP-DE		PSO-CSC		CSO	
ID	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)	ave.c	ave.T(std.T)
1	100%	4.08E+03(1.03+03)	100%	6.53E+02(8.87E+01)-	100%	7.88E+02(9.08E+01)-
2	66.7%	3.00E+05(0)	44.7%	3.00E+05(0)=	48.8%	3.00E+05(0)=
3	100%	7.90E+03(2.61E+03)	80.3%	3.00E+05(0)+	70%	3.00E+05(0)+
4	100%	6.10E+04(2.31E+04)	50.7%	3.00E+05(0)+	74.7%	3.00E+05(0)+
5	100%	7.33E+03(1.75E+03)	71.1%	2.82E+05(3.21E+04)+	81.3%	2.60E+05(7.12E+04)
6	100%	1.03E+05(4.02E+04)	94.3%	1.98E+05(2.75E+05)+	43.7%	3.00E+05(0)+
7	100%	3.53E+03(1.67E+03)	100%	2.11E+03(2.24E+03)=	92.2%	1.01E+05(1.37E+05)+
8	100%	2.73E+03(1.27E+03)	100%	1.88E+03(1.79E+03)=	96.6%	5.08E+04(8.59E+04)+
9	68.8%	3.00E+05(0)	66.7%	3.00E+05(0)=	66.7%	3.00E+05(0)=
10	60%	3.00E+05(0)	60%	3.00E+05(0)=	60%	3.00E+05(0)=
11	53.3%	3.00E+05(0)	53.3%	3.00E+05(0)=	48.3%	3.00E+05(0)=
12	100%	2.30E+04(2.84E+03)	95.7%	9.73E+04(1.26E+06)+	12%	3.00E+05(0)+
+/=/-				5/6/1		5/6/1

* ave.c: the average value of c in all paths for tested fog computing function.
* ave.T(std.T): the average and standard test case number of each algorithm in tested benchmark functions.
* +/=/-: the number of "+," "=," and "-" represent that RP-DE uses significant less, equal, and more test cases than
compared algorithm with Wilcoxon rand-sum test with $\alpha = 0.05$.

TABLE X
BENCHMARK FUNCTIONS IN OTHER OPEN-SOURCE PROGRAMS

Tested Program/Functions	ID	lines of code	number of paths	number of dimensions
bibclean-2.08				
check_ISBN	1	82	7	12
check_ISSN	2	81	7	10
gimp-2.2.4				
gimp_rgb_to_hsv	3	51	6	3
gimp_hsv_to_rgb	4	76	15	3
gimp_hwb_to_rgb	5	61	15	3
tiff-3.8.2				
TIFF_GetSourceSamples	6	37	8	2

with PSO-CSC. In the result of "+/=/-" of Table IX, compared with PSO-CSC and CSO, RP-DE uses significantly less or equal test cases in most functions. The performance of RP-DE in Table IX indicates that RP-DE is suitable for solving ATCG-PC in programs of fog computing or other research areas.

F. Experimental Finding

There are four conclusions drawn form the above experiment results. First, the parameters of F and Pc are not sensitive in RP-DE. Therefore, we select one combination of them with the best performance. Second, RP-DE significantly outperforms the DE algorithm in tested benchmark functions based on models in [14] and our proposed model. Third, RP-DE also shows advantages when it is compared with other state-of-the-art algorithms. The test-case-path relationship matrix also empowers evolutionary algorithms such as IGA, ABC, and PSO in solving ATCG-PC. Finally, compared with PSO-CSC and CSO, RP-DE is more suitable for ATCG-PC in programs of fog computing or other areas.

VI. CONCLUSION

In this paper, we propose a mathematical model to build ATCG-PC in fog computing systems as a single-objective problem. Each test case only needs to be evaluated once. This model evaluates the path coverage rate for ATCG-PC in fog computing systems. Given a maximum number of test cases, the performance of compared algorithms can be evaluated based on our proposed model, even in functions with infeasible paths. The relationship between test cases and paths is uncertain. It is difficult for classical DE to achieve high path coverage rate. A DE algorithm empowered by a test-case-path relationship matrix (RP-DE) is introduced to solve ATCG-PC in fog computing systems. The test-case-path relationship matrix collects the cor-

relation coefficients between the test cases and paths from the used test cases. The experiment results show that relationship matrix empowers evolutionary algorithms such as DE, IGA, ABC, and PSO in solving ATCG-PC. The experiment results also indicate that RP-DE is the best choice in ATCG-PC among all compared algorithms.

References

[1] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Trans. Softw. Eng.*, vol. 36, no. 6, pp. 742–762, Nov./Dec. 2010.

[2] S. Anand *et al.*, "An orchestrated survey of methodologies for automated software test case generation," *J. Syst. Softw.*, vol. 86, no. 8, pp. 1978–2001, 2013.

[3] B. Tang *et al.*, "Incorporating intelligence in fog computing for big data analysis in smart cities," *IEEE Trans. Ind. Inform.*, vol. 13, no. 5, pp. 2140–2150, Oct. 2017.

[4] P. Hu, H. Ning, T. Qiu, Y. Zhang, and X. Luo, "Fog computing based face identification and resolution scheme in internet of things," *IEEE Trans. Ind. Inform.*, vol. 13, no. 4, pp. 1910–1920, Aug. 2017.

[5] C. Zhu, V. C. Leung, L. T. Yang, and L. Shu, "Collaborative location-based sleep scheduling for wireless sensor networks integrated with mobile cloud computing," *IEEE Trans. Comput.*, vol. 64, no. 7, pp. 1844–1856, Jul. 2015.

[6] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Softw.: Pract. Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[7] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[8] T. Xie, D. Marinov, W. Schulte, and D. Notkin, "Symstra: A framework for generating object-oriented unit tests using symbolic execution," in *Proc. Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2005, pp. 365–381.

[9] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Comput. Surv.*, vol. 29, no. 4, pp. 366–427, 1997.

[10] J. R. Horgan, S. London, and M. R. Lyu, "Achieving software quality with testing coverage measures," *Comput.*, vol. 27, no. 9, pp. 60–69, 1994.

[11] G. Fraser and A. Arcuri, "Whole test suite generation," *IEEE Trans. Softw. Eng.*, vol. 39, no. 2, pp. 276–291, Feb. 2013.

[12] X. Yao, D. Gong, and W. Wang, "Test data generation for multiple paths based on local evolution," *Chin. J. Electron.*, vol. 24, no. 1, pp. 46–51, 2015.

[13] D. J. Mala, V. Mohan, and M. Kamalapriya, "Automated software test optimisation framework—An artificial bee colony optimisation-based approach," *Softw. IET*, vol. 4, no. 5, pp. 334–348, 2010.

[14] H. Huang, F. Liu, X. Zhuo, and Z. Hao, "Differential evolution based on self-adaptive fitness function for automated test case generation," *IEEE Comput. Intell. Mag.*, vol. 12, no. 2, pp. 46–55, May 2017.

[15] J. C. King, "A new approach to program testing," *ACM Sigplan Notices*, vol. 10, no. 6, pp. 228–233, 1975.

[16] M. Harman, "Software engineering meets evolutionary computation," *Comput.*, vol. 44, no. 10, pp. 31–39, 2011.

[17] L. Wang, Z. Yue, and H. Hou, "Genetic algorithms and its application in software test data generation," *J. Beijing Univ. Aeronaut. Astronaut.*, vol. 2, pp. 617–620, 1998.

[18] Y. Suresh and S. K. Rath, "A genetic algorithm based approach for test data generation in basis path testing," *Int. J. Soft Comput. Softw. Eng.*, vol. 3, no. 3, pp. 326–332, 2013.

[19] N. Zhang, B. Wu, and X. Bao, "Automatic generation of test cases based on multi-population genetic algorithm," *Int. J. Multimedia Ubiquitous Eng.*, vol. 10, no. 6, pp. 113–122, 2015.

[20] A. Bouchachia, "An immune genetic algorithm for software test data generation," in *Proc. Int. Conf. Hybrid Intell. Syst.*, 2007, pp. 84–89.

[21] J. C. Lin and P. L. Yeh, "Automatic test data generation for path testing using gas," *Inf. Sci.*, vol. 131, no. 14, pp. 47–64, 2001.

[22] D. J. Mala, M. Kamalapriya, R. Shobana, and V. Mohan, "A non-pheromone based intelligent swarm optimization technique in software test suite optimization," in *Proc. Int. Conf. Intell. Agent Multi-Agent Syst.*, 2009, pp. 1–5.

[23] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.* 2012, pp. 13–16.

[24] M. Mukherjee *et al.*, "Security and privacy in fog computing: Challenges," *IEEE Access*, vol. 5, pp. 19 293–19 304, Sep. 6, 2017.

[25] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. New York, NY, USA: Springer-Verlag New York, Inc., 2005.

[26] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE Trans. Evol. Comput.*, vol. 10, no. 6, pp. 646–657, Dec. 2006.

[27] W. Gong, Z. Cai, and D. Liang, "Adaptive ranking mutation operator based differential evolution for constrained optimization." *IEEE Trans. Cybern.*, vol. 45, no. 4, pp. 716–727, Apr. 2015.

[28] R. Ding, X. Feng, S. Li, and H. Dong, "Automatic generation of software test data based on hybrid particle swarm genetic algorithm," in *Proc. Elect. Electron. Eng.*, 2012, pp. 670–673.

[29] W. Gong and Z. Cai, "Differential evolution with ranking-based mutation operators," *IEEE Trans. Cybern.*, vol. 43, no. 6, pp. 2066–2081, Dec. 2013.

[30] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, Apr. 2009.

[31] W. Gong, Z. Cai, and Y. Wang, "Repairing the crossover rate in adaptive differential evolution," *Appl. Soft Comput.*, vol. 15, pp. 149–168, 2014.

[32] N. Mansour and M. Salame, "Data generation for path testing," *Softw. Qual. J.*, vol. 12, no. 2, pp. 121–136, 2004.

[33] M. Otto, "Pattern recognition and classification," *Chemometrics: Statist. Comput. Appl. Analytical Chemistry*, pp. 135–211, 1999.

[34] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, no. 4, pp. 341–359, 1997.

[35] D. J. Mala, V. Mohan, and M. Kamalapriya, "Automated software test optimisation frameworkAn artificial bee colony optimisation-based approach," *Softw. IET*, vol. 4, no. 5, pp. 334–348, 2010.

[36] M. R. Girgis, A. S. Ghiduk, and E. H. Abdelkawy, "Automatic data flow test paths generation using the genetical swarm optimization technique," *Int. J. Comput. Appl.*, vol. 116, no. 22, pp. 25–33, 2015.

[37] R. Cheng and Y. Jin, "A competitive swarm optimizer for large scale optimization," *IEEE Trans. Cybern.*, vol. 45, no. 2, pp. 191–204, Feb. 2015.

[38] H. Huang, L. Lv, S. Ye, and Z. Hao, "Particle swarm optimization with convergence speed controller for large-scale numerical optimization," *Soft Comput.*, pp. 1–17, 2018.

[39] P. McMinn, M. Harman, K. Lakhotia, Y. Hassoun, and J. Wegener, "Input domain reduction through irrelevant variable removal and its effect on local, global, and hybrid search-based structural test data generation," *IEEE Trans. Softw. Eng.*, vol. 38, no. 2, pp. 453–477, Mar./Apr. 2012.

[40] M. Harman and P. McMinn, "A theoretical and empirical study of search-based testing: Local, global, and hybrid search," *IEEE Trans. Softw. Eng.*, vol. 36, no. 2, pp. 226–247, Mar./Apr. 2010.

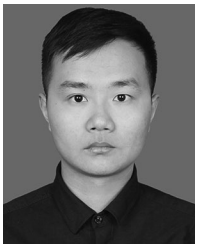
[41] R. G. Steel and J. H. Torrie, *Principle and Procedures of Statistic: A Biometrical Approach*. New York, NY, USA: McGraw-Hill, 1980.

[42] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.

Han Huang (M'15) received the B. Man. degree in information management and information system from School of Mathematics, South China University of Technology, Guangzhou, China, in 2003, and the Ph.D. degree in computer science from the South China University of Technology (SCUT), Guangzhou, China, in 2008.

Currently, he is a Professor with the School of Software Engineering in SCUT. His research interests include evolutionary computation, and swarm intelligence and their application.

Dr. Huang is a senior member of CCF and a member of IEEE.



Fangqing Liu (S'18) received the B.E. degree in software engineering from South China University of Technology, Guangzhou, China, in 2016. He is currently a working toward the postgraduate degree in software engineering at School of Software Engineering, South China University of technology, Guangzhou, China.

His research interests include automated software test case generation and evolutionary computation for software testing.

Zhongming Yang received the Information Engineering degree from Guangdong University of Technology, Guangdong, China, in 1999, and the master's degree in software engineering from Huazhong University of Science Technology, Hubei, China, in 2008.

He is currently an Associate Professor with the College of Computer Engineering Technical, Guangdong Institute of Science and Technology, Guangdong Sheng, China. His research interests include intelligent algorithm software engineering and computer network.

Zhifeng Hao (M'18) received the B.Sc. degree in mathematics from Sun Yatsen University, Guangzhou, China, in 1990, and the Ph.D. degree in mathematics from Nanjing University, Nanjing, China, in 1995.

He is currently a Professor with the School of Computer, Guangdong University of Technology, Guangzhou, China, and the School of Mathematics and Big Data, Foshan University, Foshan, China. His research interests include various aspects of algebra, machine learning, data mining, and evolutionary algorithms.