ISSN 1330-3651 (Print), ISSN 1848-6339 (Online) https://doi.org/10.17559/TV-20220130140211
Original scientific paper

Automated Test Case Generation Based on Competitive Swarm Optimizer with Schema and Node Branch Archive

Xiaohu DAI, Bin NING*, Qiong GU, Chunyang HU, Shuijia LI

Abstract: Software testing plays an important role in the software development life cycle, among which automated test case generation (ATCG) technology is widely concerned because of its low cost and high degree of automation. In the process of using search-based algorithms to solve the automated test case generation for path coverage (ATCG-PC), how to minimize the generation of redundant test cases under the premise of 100% path coverage has always been a challenge. Inspired by improving the search ability of the search-based algorithm itself and the prior knowledge in the field of ATCG-PC, we propose a competitive swarm optimizer with schema and node branch archive (SNBAr-CSO) algorithm to solve the problem of complex test case generation with multiple variables in nodes. On the basis of competitive swarm optimizer, this algorithm uses the prior knowledge of schema to find all variables that affect the direction of a node branch quickly, and uses node branch archive to record the relationship between node branch direction and variable value. The experimental results of 12 practical programs on iFogSim and CoreNLP show that compared with other newly proposed algorithms, SNBAr-CSO can greatly reduce the number of redundant test cases under the premise of 100% path coverage.

Keywords: automated test case generation; competitive swarm optimizer; node branch archive; path coverage; schema

1 INTRODUCTION

As an important part of the software life cycle, the purpose of software testing is to find possible defects in software products [1]. Functional testing and structural testing are the most common software testing activities. The focus of functional testing is to meet the needs of the program, while structural testing focuses on revealing the logical defects of the program [2]. Automated test case generation (ATCG) technology is white box testing, it belongs to a kind of structural testing.Compared with the traditional manual method, it can save a lot of human resources and find more logical defects contained in software products [3].

The main goal of ATCG is to generate a set of test cases that meet specific coverage criteria. At present, the commonly used coverage standards include branch coverage, statement coverage and path coverage. Among these coverage criteria, path coverage is considered to be the most stringent coverage criterion because it requires that each possible path of the program under test be executed at least once [4]. In particular, the test case set that meets the path coverage can also cover all branches and statements [5]. Therefore, test case generation for path coverage (ATCG-PC) has become a hot spot in the field of ATCG.

The common solution methods of ATCG-PC include random method [6], symbolic execution method [7], program instrumentation method [8] and search-based algorithms [9]. The random generation method randomly generates a large number of test cases in the space of the definition domain of problem variables. Although this method is simple to implement and easy to quickly generate a large number of test cases, the test case set generated is very large and the path coverage is low. The symbolic execution method uses symbolic expressions to replace the corresponding variables in the path as program input, and generates test cases by solving relevant constraints. Because the computational complexity of this method is exponential with the constraint expression in the measured program, it cannot solve the practical problem well. The program instrumentation method obtains the test value by inserting the test statement set into each judgment statement and loop statement in the program. Due to the direct instrumentation of the source code, this method has high accuracy and pertinence, but the instrumentation operation workload is large and pollutes the source code. The search-based algorithm transforms ATCG-PC into a combinatorial optimization problem, and guides individuals to cover all paths through the experience learned in the search process. Because search-based algorithms have few constraints on the problem to be solved and do not rely on gradient information, search-based testing has become a new trend to solve ATCG-PC.

Researchers propose a variety of methods to solve ATCG-PC based on search-based algorithms. They can be roughly classified into two different categories.

The first category focuses on improving the global or local search ability of ATCG-PC algorithms. For example, Wang et al. [10] and Suresh [11] used genetic algorithm (GA) to solve ATCG-PC. On the basis of GA, Yao et al. [12] proposed a multi-population genetic algorithm with individual sharing, so as to more efficiently generate test cases satisfying path coverage. The above algorithms focus on improving search-based algorithms themselves. In addition, the design of fitness function can help search-based algorithms find more paths. Lin [13] proposed a GA based on the fitness function of Hamming distance to generate test cases covering the target path. Huang et al. [14] used an adaptive fitness function to improve the fitness of test cases with uncovered nodes. Sahoo et al. [15] proposed a value combination branch distance function based on path coverage criterion, and combined it with particle swarm optimization (PSO) algorithm to automatically generate test cases.

The second strategy uses the specific knowledge of ATCG-PC to help reduce the search space. For example, Huang et al. [16] proposed a differential evolution with relational matrix (RP-DE). Through the relational matrix, the relationship between the dimension of test case variables and nodes can be found, thus avoiding a lot of redundant search processes. Dai [4] proposed a node branch archive (NBAr) based on the relationship matrix, which further reduced the search space. Liu et al. [5] designed a manifold inspired search algorithm (MISA), which reduced the search domain by equivalent mapping subspace. Gong et al. [17] proposed a target path grouping strategy, in which all target paths are divided into multiple

groups, and the paths belonging to the same group are highly similar to each other, so that the targets in the same group can be quickly covered.

To sum up, we note that in addition to adopting more novel search-based algorithms or designing better fitness functions, the specific knowledge in the field of ATCG-PC can better improve the efficiency of the algorithm in solving ATCG-PC. Inspired by this idea, we find that for a complex ATCG-PC problem with multiple variables in a node, if we can understand all the variables affecting the node direction in advance, we will obtain competitive performance on this kind of problem. Therefore, this paper combines the above two improvement strategies and proposes a competitive swarm optimizer based on schema and node branch archive (SNBAr-CSO). Among them, competitive swarm optimizer algorithm (CSO) [18] is a competition search-based algorithm with good search ability. Schema is a kind of prior knowledge designed for a complex ATCG-PC problem with multiple variables on a node. Node branch archive can reduce a lot of search space by recording the relationship between nodes and variables.

The main contributions of our work are as follows:

- Aiming at the complex ATCG-PC problem with multiple variables in nodes, this paper deeply analyzes and puts forward a priori knowledge of schema, which is helpful to quickly find out all variables affecting nodes.
- A strategy combining schema and node branch archive (SNBAr) is proposed. Schema and node branch archive can efficiently find the relationship between test case variable dimensions and nodes. The combination of these two strategies can reduce a lot of unnecessary search space.
- Competitive swarm optimizer with schema and node branch archive (SNBAr-CSO) is proposed. The SNBAr-CSO is compared with other state-of-the-art algorithms, and the experimental results show that the performance of our algorithm is better than the latest solution of ATCG-PC.

The rest of this paper is organized as follows. Section 2 introduces the preliminary knowledge of search-based algorithms for solving ATCG-PC. Section 3 introduces CSO and NBAr and analyzes the advantages and improvements of NBAr. Section 4 introduces the details and overall process of SNBAr-CSO. In Section 5, the comparison between SNBAr-CSO and other algorithms and the separation experiments under different strategies are discussed. A brief summary of this paper is presented in Section 6.

2 BACKGROUND

This section introduces the preliminary knowledge of search-based algorithms for solving ATCG-PC, including basic concepts, mathematical model, fitness function and several common search-based algorithms.

2.1 Basic Concepts of ATCG-PC

According to the definition of [19], there are some basic concepts for solving ATCG-PC.

Definition 1 (control flow graph): Control flow graph (CFG) is a directed graph $G = (N, E, s, e)$. It can concisely represent the path structure of the program under test. where N represents the node set, the edge set $E = \{< n_1, n_2 >| n_1, n_2 \in N\}$ represents an edge from node n_1 to n_2, s and e are the start and end nodes of the program respectively.

Definition 2 (Node): Node represents the basic block in the program under test, and the statements in the basic block are executed sequentially. A conditional node is a node that contains predicates, and each of its output edges is called a branch.

Definition 3 (test case): A test case is a vector $X_i = (x_{i1}, x_{i2}, \ldots, x_{in})$ with n dimensions, in which the dimension is composed of variables in the program under test.

Definition 4 (path): A path $p_i <s, n_1, \ldots, n_k, e>$ is a sequence composed of at least start node and end node, which represents the actual execution of the test case in the program under test.

Definition 5 (path coverage): If the path passed by test case X_i in the program under test is p_j, X_i covers the path p_j.

According to the above definition, the path can be represented by a sequence of nodes. Determining which condition nodes the path passes through and the branch direction on these condition nodes can uniquely determine the path. In order to more concisely represent a specific path, we only use conditional nodes to encode it, in which non null characters indicate that the test case passes through the node, null characters indicate that the test case skips the node, and specific branch directions can be represented by characters 0-9. For the real variable in the program to be tested, its corresponding value can be directly encoded. String variables can be represented by ASCII codes composed of corresponding characters.

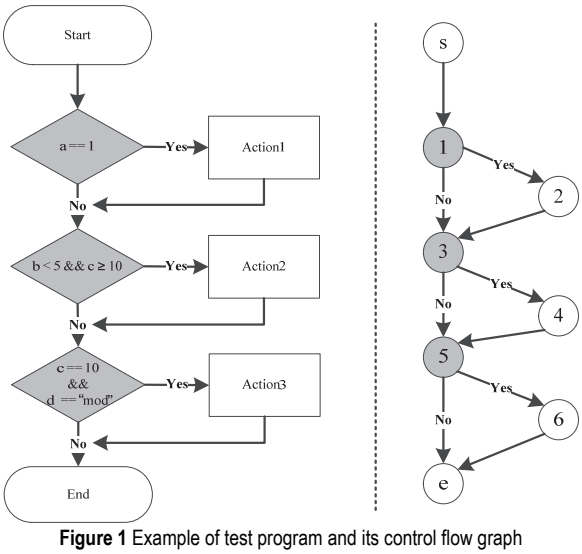

Figure 1 Example of test program and its control flow graph

As shown in Fig. 1, the program includes 4 variables (a, b, c, d where a, b, c, belong to real variables and d belongs to string variables) and 8 nodes (s, e are start and end nodes, ②, ④, ⑥ belong to ordinary nodes and ①, ③, ⑤ belong to branch nodes). We use '0' and '1' to represent "No" and "Yes" branches of the branch node respectively. For path <s, 1, 2, 3, 4, 5, 6, e>, its condition nodes are ①, ③, ⑤, and the branch directions on these condition nodes are all "Yes". Therefore, its path code can be expressed as "111". If the test case X_i needs to cover this path, its variable value should be $a = 1$, $b = 4$, $c = 10$, $d =$

"mod", and the corresponding variable code of X_i = (1, 4, 10, 109, 111, 100).

2.2 Mathematical Model of ATCG-PC

In order to better describe the ATCG-PC problem, researchers have proposed a variety of mathematical models. Fraser et al. [20] proposed a test suite model to minimize the redundant test cases under the condition of meeting the coverage criteria. Huang et al. [14] designed an adaptive evaluation model that is conducive to including the uncovered nodes. Because models [14] and [20] cover all target paths at the same time, these models are not helpful when it is necessary to cover a specific target path. In addition, ATCG-PC is also modelled as a multi-objective problem. Mala et al. [21] established a mathematical model with two objectives, which considers both maximizing the number of path coverage and minimizing the number of test cases. In the multi-objective model, each test case needs to evaluate all the remaining uncovered paths, which leads to a lot of function evaluations consumption.

The mathematical model in this paper is based on [16]. The model considers both the target path coverage problem and the problem including the path that cannot be covered. Its solution goal is to maximize the path coverage c under the condition of the maximum acceptable number of testcases Mcn. The specific description of the model is as follows:

$$c = \frac{l}{L} \times 100\% \tag{1}$$

$$l = \sum_{j=1}^{L} \min\left\{1, \sum_{i=1}^{T} \Theta_{i,j}\right\} \tag{2}$$

$$\Theta_{i,j} = \begin{cases} 1, & \text{If test case } X_i \text{ covers path } p_j \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$T \le Mcn \tag{4}$$

Eq. (1) represents the calculation method of path coverage rate c, where l represents the number of paths covered by the generated test case, L is the total number of paths in the program under test. Eq. (2) and Eq. (3) define the calculation method of l, where $\sum_{i=1}^{T} \Theta_{i,j}$ counts whether the generated test cases cover the path p_j, and when multiple test cases cover the same path, the value of l is accumulated by one. Eq. (4) represents the maximum acceptable number of test cases generated T.

2.3 Fitness Function

When the search-based algorithms are used to solve ATCG-PC, the fitness function is used to evaluate the quality of the generated test cases. By comparing the fitness values, the appropriate individuals are selected as the offspring in each iteration, so as to effectively guide the algorithm to find the test case set covering all paths. Based on the selected mathematical model, this paper uses the branch distance method [14] as the fitness function, which is widely used in ATCG-PC. The calculation method is as follows:

$$fitness(X_i) = \sum_{j=1}^{m} f\left(p_{X_i}, j\right) \tag{5}$$

$$f\left(p_{X_i}, j\right) = \frac{1}{BD\left(X_i^j\right) + \varepsilon} \tag{6}$$

where m represents the number of nodes contained in the program under test, $fitness(X_i)$ is the sum of fitness at all nodes, $f\left(p_{X_i}, j\right)$ represents the fitness at j-th node, $BD\left(X_i^j\right)$ reflects the deviation between the actual branch predicate and the required branch predicate of the test case X_i on the j-th node, ε is a constant to avoid divisor 0. Tab. 1 lists the commonly used branch predicates and their corresponding branch distance function calculation methods.

Table 1 The calculation of branch distance function

No.	Predicate	Branch distance
1	Boolean	If true then 0 else K
2	$A > B$	If $(B - A) < 0$ then 0 else $(B - A) + K$
3	$A \ge B$	If $(B - A) \le 0$ then 0 else $(B - A) + K$
4	$A < B$	If $(A - B) < 0$ then 0 else $(A - B) + K$
5	$A \le B$	If $(A - B) \le 0$ then 0 else $(A - B) + K$
6	$A = B$	If $\lvert A - B \rvert = 0$ then 0 else $\lvert A - B \rvert + K$
7	$A \ne B$	If $\lvert A - B \rvert \ne 0$ then 0 else K
8	$A \wedge B$	$BD(A) + BD(B)$
9	$A \vee B$	$\mathrm{Min}(BD(A), BD(B))$

2.4 Search-Based Algorithms for ATCG-PC

This section discusses the effectiveness of different search-based algorithms, such as differential evolution (DE), [22], immune genetic algorithm (IGA), [23], artificial bee colony (ABC) [24], particle swarm optimization (PSO) [24] and competitive swarm optimizer (CSO), [18].

Search-based algorithms are a mature global optimization method. Because of its high robustness and wide applicability, it is widely used to solve ATCG-PC problems. DE generates new candidate solutions through the differential and crossover operations of individuals in the parent population. IGA combines the advantages of immune theory and basic genetic algorithm. It not only retains the search characteristics of GA, but also makes use of the multi-mechanism of immune algorithm to find the optimal solution of multi-objective function. ABC is composed of three basic elements: food source, employed bees and unemployed bees. It finds the global optimal value through the local optimization behaviour of individual artificial bees. PSO guides the population evolution by recording the global optimal solution and the optimal solution of all individuals. Different from PSO which records individual optimal solution and global optimal solution, CSO introduces a pair of competition

mechanisms, in which the particles that lose the competition will update their position by learning from the winner, so as to achieve a good balance between exploration and development. The operation process of CSO will be described in detail in Section 3.

3 COMPETITIVE SWARM OPTIMIZER AND NODE BRANCH ARCHIVE

In this section, we will describe the operation process of CSO algorithm and node branch archive in detail.

3.1 Competitive Swarm Optimizer

The competitive swarm optimizer (CSO) [17] is a group optimization algorithm for large-scale optimization. Based on the particle swarm optimization algorithm, the algorithm introduces the pairwise competition mechanism. The failed particles update their positions by learning from the winner. CSO algorithm is used in a large number of practical problems because of its simple structure and good balance between exploration and development.

In the CSO algorithm, for each particle in generation t, its position and velocity are expressed as:

$$\begin{aligned} &X_i(t)=\left(x_{i,1}(t),x_{i,2}(t),...,x_{i,n}(t)\right)\\ &V_i(t)=\left(v_{i,1}(t),v_{i,2}(t),...,v_{i,n}(t)\right)\\ &i=1,2,...,\text{pop} \end{aligned} \tag{7}$$

where i represents the population size, n is the dimension of each individual variable and pop represents the individual size of a population.

3.1.1 Initialization

During the initialization of CSO algorithm, each individual in the population will be assigned a position of random value in the definition domain, and the initial velocity of all particles is 0. The specific operation is shown in Eq. (8).

$$\begin{cases} x_{i,j}=ub_j+\text{rand}(0,1)\cdot\left(ub_j-lb_j\right)\\ v_{i,j}=0 \end{cases} \tag{8}$$

where rand[0, 1] represents the random number in the range of 0 to 1, and ub_j and lb_j represent the upper and lower limits of the variable on the j-th dimension.

3.1.2 Learning Operator

In order to make the algorithm converge to the global optimal solution, CSO introduces the pairwise competition mechanism in the population iteration process. The individuals in the group will be divided into k pairs, k = 1, 2, …, pop/2, and the individual in the same pair will compete. The losers in the competition will update their speed and position by learning from the winners. The formula is as follows:

$$\begin{aligned} V_{l,k}(t+1)=&R_1(k,t)V_{l,k}(t)+\\ &+R_2(k,t)\left(X_{w,k}(t)-X_{l,k}(t)\right)+\\ &+\varphi R_3(k,t)\left(\bar{X}_k(t)-X_{l,k}(t)\right) \end{aligned} \tag{9}$$

$$X_{l,k}(t+1)=X_{l,k}(t)+V_{l,k}(t+1) \tag{10}$$

where l and w respectively represent the individuals who lose or win in the competition, R_1, R_2, and R_3, are random numbers between (0, 1), $\bar{X}_k$ means the average position value of relevant particles, and φ is the parameter controlling the influence of $\bar{X}_k$.

3.2 Node Branch Archive

The node branch archive (NBAr) can record the relationship between the dimension of test case variables and the node branch direction in the search process, and reduce a lot of search space through this relationship. Next, we will further introduce the specific process of NBAr and put forward some improvement suggestions for its existing problems.

In the process of test case generation, variables with specific values can make the test case pass through a specific direction on a node. As shown in Fig. 1, node a == 1 is only related to variable a, and when the variable value is a = 1 the direction of the node is "Yes". Therefore, if the direct relationship between variable values and nodes can be found, the search for irrelevant dimensions can be avoided.

NBAr uses two matrices A and R to record the relationship between variables and nodes found in the search process. The matrix R can record the correlation between variables and nodes, that is, which variables determine which nodes directions, thus reducing the search process for unnecessary dimensions. The matrix A records the relationship between the variable value and the node direction, that is, what value should be taken when the node is in a specific direction. If the variables that most affect a node can be found by matrix R, we can try to find the value of the variables that affect the direction of the node by matrix A.

NBAr mainly records the relationship between test cases and nodes through two matrices to reduce the search space. However, the strategy can only find one variable value that affects the direction of a node at a time, so it cannot efficiently solve complex ATCG-PC problems where the direction of a node is determined by multiple variables. If we can find all variables and their values that affect the direction of the node at one time, we can greatly reduce the search process of irrelevant variables and variable values.

4 OUR APPROACH

In this section, a competitive swarm optimizer with schema and node branch archive (SNBAr-CSO) will be proposed. We will introduce SNBAr in detail and the whole process of applying SNBAr to CSO. Finally, an example is given to illustrate how SNBAr-CSO can solve

the multivariable ATCG-PC problem. The variables used in SNBAr-CSO are shown in Tab. 2.

Table 2 Notation of basic variables in SNBAr

Variable name	Description
L	The total number of paths in the program under test
X_i	The optimized test case
S_i	The schema corresponding to the ith node
p_{target}	The selected target path
p_{Xi}^{j}	The jth character in the path encoding string of p_{Xi}
n	The number of variable dimensions
m	The total number of nodes
pop	The individual size of a population
dim	The selected optimized dimension

4.1 Schema

As shown in Section 3, node branch archive can reduce a lot of search space by recording the relationship between test case variables and node branch direction. However, NBAr can only find one variable that affects the direction of node branch at most each time, which is not conducive to solving programs with multiple variables in nodes. Therefore, we propose a priori knowledge of schema. The combination of schema and node branch archive can better solve the complex ATCG-PC problem with multiple variables in nodes.

Schema is a set of all variables that affect the branch direction of the same node. For a program to be tested, its schemas set can be expressed as:

$$S = \{S_1, S_2, ..., S_m\} \tag{11}$$

where m represents the number of nodes contained in the program. Each schema corresponds to each branch node in the program under test. For the i-th node, the schema of this node can be expressed as:

$$S_i = \{S_{i1}, S_{i2}, ..., S_{ik}\} \tag{12}$$

where k is the number of variables contained in S_i. The value of these variables will affect the branch direction of the node. We can directly find all the variables that affect the direction of the node through the schema of the corresponding node. The schema of the program shown in Fig. 1 is shown in Tab. 3.

Table 3 Schema in example program

Schema	Branch node	Included variables
s_1	node①	a
s_2	node③	b, c
s_3	node⑤	c, d

4.2 Dynamic Target Path Selection Strategy

When solving the ATCG-PC single path coverage problem, the search-based algorithm covers the uncovered paths one by one through multiple iterations, so the selection order of target paths will affect the solution performance of the algorithm. In order to select a more suitable target path, we adopt a dynamic target path selection strategy based on similarity.

For each individual X_i in the population, we choose the most suitable target path p_{target} from the paths without coverage. Compared with other paths, the path coding of p_{X_i} is the closest to p_{target}. Algorithm 1 shows the selection process of p_{target}. We use w to record the similarity between p_{X_i} and each path. If p_j has been covered, w_j is assigned to 0. If p_j is not covered, w_j records the number of similar codes in p_{X_i} and p_j path codes. The higher the similarity is corresponds to the path closest to p_{X_i}. When all the paths are compared, we use roulette-wheel selection based on w to select the closest path as p_{target}.

Algorithm 1: Dynamic target path selection

```
Input: The optimized test case X_i
Output: The selected target path p_target
for each j ∈ {1, 2, ..., L} do
    Initialize w to zero matrix.
    if p_j is covered covered then
        w_j ← 0;
    end
    else
        for each k ∈ {1, 2, ..., m} do
            if p^k_{X_i} == p^k_j then
                w_j ← w_j + 1;
            end
        end
    end
end
Use roulette-wheel selection to get p_target based on w;
```

4.3 Search Process of SNBAr

As shown in Algorithm 2, the overall search process of SNBAr can be roughly divided into two parts: (1) search based on schema and matrix A; (2) search based on scatter search strategy.

Algorithm 2: The searching process of SNBAr

```
Input: The population of the test cases X.
Output: Test case set covering all paths.
1  Initialize the node branch archive matrix A to null matrix.
2  for each i ∈ {1, 2, ..., pop} do
3      Dynamic selection of target path p_target for x_i based on
       Algorithm 1 .
4      for each j ∈ {1, 2, ..., m} do
5          if p^j_{X_i} ≠ p^j_target then
6              Select a dimension dim from S_j.
7              if A[dim][j][p^j_target] ≠ null then
8                  for each k ∈ S_j do
9                      X_{i,j} ← A[k][j][p^j_target];
10                 end
11             end
12             else
13                 for each k ∈ S_j do
14                     Initialization search step length:
                       step ← (ub_k − lb_k)/2
15                     while p_target is not covered and step > 0
                       do
16                         Using discrete search algorithm to
                           search.
17                         When a new path is covered,
                           Algorithm 3 is used to update the
                           matrix A.
18                     end
19                     if p_target is covered then
20                         Go back to Line 2.
21                     end
22                 end
23             end
24         end
25     end
26 end
```

After the p_{target} is determined through Algorithm 1, we compare each value of the path codes of p_{X_i} and p_{target}. As shown in line 5 of Algorithm 2, if the values of p_{X_i} and p_{target} are different on *j*-th node, it means that the directions of p_{X_i} and p_{target} on *j*-th node are different. As shown in line 7 - 11 of Algorithm 2, we judge whether matrix A has recorded the value of the variable that can make the direction of X_i on *j*-th node the same as that of p_{target}. If the values of these variables can be found, the values of the variables contained in S_j will be assigned X_i. In this way, we can reduce the search space and a lot of fitness evaluations consumption by using the schema and the relationship between variables and nodes.

When the schema and matrix A cannot be used to reduce the search space, we use scatter search strategy [26] to search each variable in S_j. The scatter search strategy can select the location of each search according to fitness, and gradually reduce the search step to find the individual with the best fitness.

In order to ensure the effectiveness of matrix A, every time a new path is covered in the search process, the matrix A needs to be updated according to Algorithm 3.

Algorithm 3: To update the A matrix

Input: The node branch archive matrix A, the optimized test case X_i, the offspring test case X_{new}
Output: The updated A matrix

```
1 for each j ∈ {1, 2, ..., m} do
2     if p^j_{X_i} ≠ p^j_{X_new} then
3         for each k ∈ S_j do
4             A[k][j][p^j_{X_k}] ← X_{k,new};
5         end
6     end
7 end
```

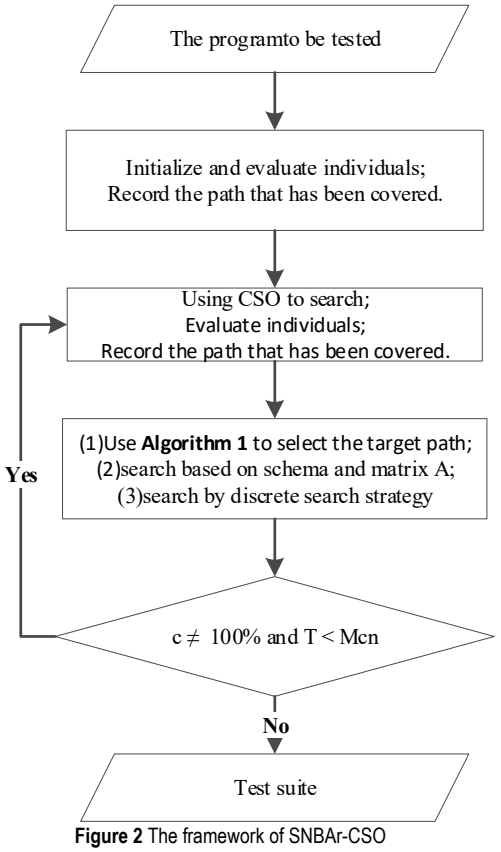

Figure 2 The framework of SNBAr-CSO

4.4 The Framework of SNBAr-CSO

As mentioned in Section 2, search-based algorithms are widely used to solve ATCG-PC problems. In this paper, SNBAr is applied to CSO, and the overall process of SNBAr-CSO is shown in Fig. 2.

Firstly, each individual in the population is initialized randomly within the value range of each variable to form the initial population, and then evaluate each individual through fitness function, and record the path covered at this time.

When the initial population is generated, the algorithm goes into the global and local search stages. In the global search stage, CSO is used to guide the whole population to generate a new offspring population, and the individuals with better fitness in the offspring population will be retained. The purpose of the local search phase is to generate test cases covering the target path. In this phase, SNBAr is used to reduce redundant search process. When SNBAr fails, scatter search strategy is used to search.

SNBAr-CSO combines the global search domain with local search. On the basis of fitness function, CSO is used to make the whole population select the individuals with the largest number of uncovered nodes. At the same time, local search strategy is used to solve the coverage problem of the target path.

4.5 The Illustrative Examples for SNBAr

We use the function in Fig. 2 to show the search process of SNBAr. According to the basic definition of ATCG-PC in Section 2, all paths contained in this function are $P = \{p_1, p_2, \ldots, p_8\}$, where p_1 = "000", p_2 = "001", p_3 = "010", p_4 = "011", p_5 = "100", p_6 = "101", p_7 = "110", p_8 = "111". $S = \{S_1, S_2, S_3\}$, the related variables of each schema are $S_1 = \{a\}$, $S_2 = \{b, c\}$, $S_3 = \{c, d\}$.

Suppose that after initializing the population and CSO operation, the covered path is p_1, and the remaining uncovered path set is $P_{remind} = \{p_1, p_2, \ldots, p_8\}$. In this case, the test case to be optimized is X_i = (3, 5, 10, 109, 117, 103). According to Algorithm 1, the target path p_{target} is selected as p_2. Because the direction of path p_1 covered by p_{X_i} is different from that of p_{target} on node⑤, our goal is to make X_i cover p_2 by changing the values of variables that affect the direction of node⑤. Since matrix A is null at this time, scatter search strategy is used on the variable dimension contained in S_3. If the new offspring in the search process covers p_2, matrix A is updated according to Algorithm 3. Since the branch direction of p_2 on node⑤ is "Yes", the updated matrix A contains the values of all variables that make the branch direction of node⑤ is "Yes".

In the next search process, if X_i = (1, 4, 10, 103, 110, 123), the path p_{X_i} covers is p_7, and the target path is p_8, then the direction of p_{X_i} and p_{target} on node⑤ is different. Since matrix A has recorded the value of the variable that makes node⑤ direction "Yes" in the previous search process, matrix A can directly assign values of variables contained in S_3 to X_i, so that X_i directly covers p_8.

Table 4 Benchmark programs in iFogSim and CoreNLP

No.	Type	Program	Line	Dimension	Path	Search space	Probability of covering the most difficult path
1	iFogSim	Transmit	30	3	2	1.68E+07	5.96E–08
2		Send	47	2	9	4.00E+12	0
3		ProcessEvent	67	7	9	6.00E+19	1.25E–06
4		ExecuteTuple	41	7	5	8.44E+14	5.33E–15
5		CheckCloudletCompletion	43	5	6	6.71E+07	4.47E–08
6		GetResultantTuple	73	8	7	1.69E+15	1.78E–15
7	CoreNLP	InitFactory	86	7	48	6.04E+23	1.65E–24
8		CleanXmlAnnotator	21	6	3	1.84E+19	3.55E–15
9		WordsToSentenceAnnotator	108	11	12	1.89E+22	2.33E–10
10		Annotate	30	4	3	3.60E+16	8.33E–17
11		NerClassifierCombiner	30	11	4	1.44E+17	3.55E–15
12		SetTrueCaseText	37	6	10	2.20E+12	9.07E–13

Note: "Search space" means the size of the decision space of the benchmark function.
"Probability of the most difficult path" means the probability of covering the most difficult path with randomly generated test cases.

5 EXPERIMENT AND RESULT ANALYSIS

In this section, we will verify the performance of the proposed SNBAr-CSO through comprehensive experiments. Firstly, the basic settings of the experiment, including benchmark programs, evaluation indicators, experimental environment and parameter setting will be introduced, and then the advantages of SNBAr-CSO will be proved through three experiments.

5.1 Experimental Setup

In order to better verify the performance of SNBAr-CSO, we selected 12 benchmark programs from iFogSim [27] and CoreNLP [28]. Among them, iFogSim is a simulation development kit, which uses the mode of sensor, process, driver and distributed data flow to simulate the application scenario in fog computing environment. CoreNLP is a set of natural language analysis tools written in Java provided by Stanford. These 12 benchmark programs have been used by many researchers such as Dai [4], Liu [5]and Huang [14] in the experimental study of ATCG-PC. The detailed information of these benchmark programs is shown in Tab. 4.

In the experiment, the following performance indicators were used to compare the performance of different algorithms.

1) *Ave.T*: The average of test cases generated in 30 independent executions. The smaller the average value, the higher the efficiency of the algorithm.

2) *Ave.c*: Average path coverage rate of 30 independent executions. The higher the path coverage, the more paths the algorithm covers.

This paper designed a total of three groups of experiments, the specific content of which is as follows:

1) SNBAr-CSO will be compared with some newly proposed algorithms. Specifically, these algorithms are GPE-IS [29], MISA [5], NBAr-DE [4] and RP-DE [16].

2) SNBAr-CSO will be compared with DE [14], IGA [30], ABC [21], PSO [25] and their variants SNBAr-DE, SNBAr-IGA, SNBAr-ABC, SNBAr-PSO.

3) The separation experiments of SNBAr-CSO under different strategies will be discussed.

The purpose of the first experiment is to discuss the performance of SNBAr-CSO and the newly proposed algorithms. The second experiment is to analyze whether the SNBAr improves the performance of search-based algorithms for solving ATCG-PC problems and which search-based algorithms perform better in combination with SNBAr. The third experiment is to discuss the performance of SNBAr-CSO in the case of different strategies chosen.

Table 5 Experimental parameter Setting

Algorithm	Parameter	Value
ALL	Population size pop	50
	The maximum acceptable number of test cases generated *Mcn*	3.00E+05
	Number of runs	30
CSO	φ	0.8
DE	Factor parameter F	0.5
	Crossover probability P_c	0.2
IGA	Crossover probability	0.8
	Mutation probability	0.1
ABC	Limit	2
	Bee number	50
PSO	c_1	1.5
	c_2	2
	Weight value w	0.4
GPE-IS	γ	$0.005\cdot(up_i - low_i)$

To make the results of the experiment more convincing, all the algorithms were run in the same experimental environment. The experimental computer was configured with Intel i7-7700 3.60 GHz, 16 GB, and Windows 10 OS. The population size in the experiment was set to pop = 50, and the maximum number of test case generations introduced was *Mnc* = 3.00E+05, with significance testing based on Wilcoxon rank-sum testing [31] with $\alpha = 0.05$. Experimental data were averaged from 30 independent runs. The values of the parameters in the compared algorithms are all the same as in the references [4, 5, 14, 21, 25, 29, 30], and their specific values are shown in Tab. 5.

5.2 Comparison with NBAr-DE and Some Other Algorithms

In order to verify the efficiency of our proposed SNBAr-CSO, we compare SNBAr-CSO with the latest algorithms GPE-IS, MISA, NBAr-DE and RP-DE. The experimental results are shown in Tab. 6. The best results among SNBAr-CSO, GPE-IS, MISA, NBAr-DE and RP-DE are highlighted in bold. These symbols "+/=/–" indicate that our method is superior to, equal to or inferior to GPE-IS, MISA, NBAr-DE and RP-DE, respectively. These marks have the same meaning in other tables.

As shown in Tab. 6, because the benchmark function No. 2 contains unfeasible path, all algorithms cover only

67% of the paths. The results of SNBAr-CSO on benchmark programs No. 1, No. 4, No. 5, No. 6, No. 7, No. 8, No. 9 and No. 12 are better than GPE-IS. Only the test case consumption of No. 10 is greater than GPE-IS. Compared with MISA, except that the test case consumption of SNBAr-CSO and MISA on benchmark programs No. 1 and No. 5 is similar, the statistical results of SNBAr-CSO on most benchmark functions are better than that of MISA. On benchmark programs No. 9 and No. 12, the test case consumption of SNBAr-CSO is one order of magnitude less than that of MISA.

In the comparison between SNBAr-CSO, NBAr-DE and RP-DE, the number of test cases generated by SNBAr-CSO is significantly less than that of NBAr-DE and RP-DE. SNBAr-CSO maintains 100% path coverage on all test functions, while NBAr-DE covers only a few paths on benchmark functions with No. 9, No. 11 and No. 12, RP-DE covers only 18% of the path on No. 12. This is because these three benchmark functions contain variables of multiple dimensions, and they contain many nodes composed of multiple variables, so the search space is very large. It can be seen that the introduction of schema reduces a lot of search space.

In conclusion, compared with GPE-IS, MISA, NBAr-DE and RP-DE, SNBAr-CSO can cover all paths with less test case consumption. Its advantage lies in the combination with schema and NBAr to quickly find all variables and their values that affect the direction of nodes, which can reduce a lot of search space. Compared with MISA and NBAr-DE, SNBAr-CSO is more suitable for solving complex multivariable ATCG-PC problems.

Table 6 Experimental results of SNBAr-CSO and some other algorithms

No.	SNBAr-CSO		GPE-IS			MISA			NBAr-DE			RP-DE		
	Ave.T	*Ave.c*	*Ave.T*		*Ave.c*	*Ave.T*		*Ave.c*	*Ave.T*		*Ave.c*	*Ave.T*		*Ave.c*
1	**1.34E+02**	**100%**	1.88E+02	(+)	**100%**	1.04E+02	(=)	**100%**	4.33E+03	(+)	**100%**	4.46E+03	(+)	**100%**
2	**3.00E+05**	**67%**	**3.00E+05**	(=)	**67%**	**3.00E+05**	(=)	**67%**	**3.00E+05**	(=)	**67%**	**3.00E+05**	(=)	**67%**
3	**3.70E+02**	**100%**	3.91E+02	(=)	**100%**	1.96E+03	(+)	**100%**	4.73E+03	(+)	**100%**	5.54E+03	(+)	**100%**
4	**2.19E+02**	**100%**	3.57E+02	(+)	**100%**	3.20E+02	(+)	**100%**	2.19E+04	(+)	**100%**	3.82E+04	(+)	**100%**
5	**1.49E+02**	**100%**	1.95E+02	(+)	**100%**	1.59E+02	(=)	**100%**	6.89E+03	(+)	**100%**	9.59E+03	(+)	**100%**
6	**2.35E+02**	**100%**	3.34E+02	(+)	**100%**	9.30E+02	(+)	**100%**	3.49E+04	(+)	**100%**	1.62E+05	(+)	97%
7	**1.43E+03**	**100%**	1.05E+04	(+)	**100%**	1.60E+04	(+)	**100%**	4.97E+04	(+)	**100%**	2.96E+05	(+)	78%
8	**2.05E+02**	**100%**	2.59E+02	(+)	**100%**	8.38E+02	(+)	**100%**	2.00E+04	(+)	**100%**	4.08E+04	(+)	**100%**
9	**3.56E+02**	**100%**	9.01E+02	(+)	**100%**	6.14E+03	(+)	**100%**	3.00E+05	(+)	16%	3.00E+05	(+)	16%
10	3.24E+02	**100%**	**2.74E+02**	(–)	**100%**	8.91E+02	(+)	**100%**	1.32E+04	(+)	**100%**	2.08E+04	(+)	**100%**
11	**2.40E+02**	**100%**	2.49E+02	(=)	**100%**	1.55E+03	(+)	**100%**	3.00E+05	(+)	25%	3.00E+05	(+)	25%
12	**1.43E+03**	**100%**	5.06E+03	(+)	**100%**	3.44E+03	(+)	**100%**	3.00E+05	(+)	22%	3.00E+05	(+)	18%
+/=/–			8/3/1			9/3/0			11/1/0			11/1/0		

Table 7 Experimental results of SNBAr-CSO with DE, SNBAr-DE, IGA and SNBAr-IGA

No.	SNBAr-CSO		DE			SNBAr-DE			IGA			SNBAr-IGA		
	Ave.T	*Ave.c*	*Ave.T*		*Ave.c*	*Ave.T*		*Ave.c*	*Ave.T*		*Ave.c*	*Ave.T*		*Ave.c*
1	**1.34E+02**	**100%**	4.14E+03	(+)	**100%**	1.60E+02	(+)	**100%**	1.31E+04	(+)	**100%**	**1.33E+02**	(=)	**100%**
2	**3.00E+05**	**67%**	**3.00E+05**	(=)	57%	**3.00E+05**	(=)	**67%**	**3.00E+05**	(=)	33%	**3.00E+05**	(=)	**67%**
3	**3.70E+02**	**100%**	3.00E+05	(+)	79%	4.29E+02	(+)	**100%**	3.00E+05	(+)	72%	3.91E+02	(=)	**100%**
4	**2.19E+02**	**100%**	3.00E+05	(+)	80%	2.48E+02	(+)	**100%**	2.18E+05	(+)	87%	2.38E+02	(+)	**100%**
5	1.49E+02	**100%**	4.67E+04	(+)	99%	1.74E+02	(+)	**100%**	2.66E+04	(+)	**100%**	**1.48E+02**	(–)	**100%**
6	2.35E+02	**100%**	3.00E+05	(+)	40%	2.79E+02	(+)	**100%**	3.00E+05	(+)	14%	**2.32E+02**	(–)	**100%**
7	**1.43E+03**	**100%**	3.00E+05	(+)	9%	1.50E+03	(=)	**100%**	3.00E+05	(+)	4%	1.53E+03	(+)	**100%**
8	**2.05E+02**	**100%**	2.50E+05	(+)	74%	2.30E+02	(+)	**100%**	2.82E+05	(+)	62%	2.05E+02	(=)	**100%**
9	**3.56E+02**	**100%**	3.00E+05	(+)	16%	3.78E+02	(+)	**100%**	3.00E+05	(+)	72%	9.02E+03	(+)	**100%**
10	**3.24E+02**	**100%**	3.00E+05	(+)	66%	3.44E+02	(=)	**100%**	3.00E+05	(+)	66%	8.26E+02	(=)	**100%**
11	**2.40E+02**	**100%**	3.00E+05	(+)	50%	2.67E+02	(+)	**100%**	1.14E+05	(+)	99%	2.75E+02	(=)	**100%**
12	**1.43E+03**	**100%**	3.00E+05	(+)	50%	1.54E+03	(=)	**100%**	3.00E+05	(+)	72%	1.25E+04	(+)	**100%**
+/=/–			11/1/0			8/4/0			11/1/0			4/6/2		

5.3 Comparison with Other Search-Based Algorithms and Their Variants

In this section, we apply SNBAr to different search-based algorithms, and verify the effectiveness of SNBAr strategy by comparing DE and SNBAr-DE, IGA and SNBAr-IGA, ABC and SNBAr-ABC, PSO and SNBAr-PSO. At the same time, in order to show that the combination of SNBAr and CSO has better results, we compare SNBAr-CSO with other variants based on search-based algorithms. The results are shown in Tab. 7, Tab. 8 and Fig. 3.

As shown in Tab. 7, DE only achieves 100% path coverage on benchmark function No. 1, while it only covers part of the path on most other benchmark functions. This is due to the lack of coverage of target path in local search phase. It is difficult to find test case set satisfying path coverage in large search space only through search-based algorithm. When SNBAr is combined with DE, SNBAr-DE achieves 100% path coverage on 11 benchmark functions, and the number of test cases generated is greatly reduced. Similarly, the combination of SNBAr with IGA, ABC and PSO also improves the efficiency of test case generation.

By comparing SNBAr-CSO with the variant based on the combination of search algorithms and SNBAr, we can see that SNBAr-CSO has the best statistical results on most benchmark functions from Tab. 7 and Tab. 8. In the comparison between SNBAr-CSO and SNBAr-ABC, SNBAr-CSO achieved better results in all programs except No. 2 program. In the box plots shown in Fig. 3, it can be seen intuitively that SNBAr-CSO consumes less test cases than other algorithms, and the results obtained are more concentrated. This shows that CSO can combine SNBAr strategy better than other search-based algorithms.

Table 8 Experimental results of SNBAr-CSO with ABC, SNBAr-ABC, PSO and SNBAr-PSO

No.	SNBAr-CSO		ABC			SNBAr-ABC			PSO			SNBAr-PSO		
	Ave.T	*Ave.c*	*Ave.T*		*Ave.c*	*Ave.T*		*Ave.c*	*Ave.T*		*Ave.c*	*Ave.T*		*Ave.c*
1	**1.34E+02**	**100%**	2.84E+05	(+)	53%	2.19E+02	(+)	**100%**	2.93E+05	(+)	53%	1.61E+02	(+)	**100%**
2	**3.00E+05**	**67%**	**3.00E+05**	(=)	35%	**3.00E+05**	(=)	**67%**	**3.00E+05**	(=)	44%	**3.00E+05**	(=)	**67%**
3	**3.70E+02**	**100%**	2.81E+05	(+)	87%	7.54E+02	(+)	**100%**	2.89E+05	(+)	86%	5.64E+02	(+)	**100%**
4	**2.19E+02**	**100%**	3.00E+05	(+)	41%	3.36E+02	(+)	**100%**	3.00E+05	(+)	40%	2.60E+02	(+)	**100%**
5	**1.49E+02**	**100%**	3.00E+05	(+)	67%	2.52E+02	(+)	**100%**	3.00E+05	(+)	67%	1.77E+02	(+)	**100%**
6	**2.35E+02**	**100%**	3.00E+05	(+)	14%	1.08E+04	(+)	**100%**	3.00E+05	(+)	14%	1.22E+05	(+)	94%
7	**1.43E+03**	**100%**	3.00E+05	(+)	16%	8.81E+03	(+)	**100%**	3.00E+05	(+)	4%	3.00E+05	(+)	85%
8	**2.05E+02**	**100%**	2.91E+05	(+)	52%	2.88E+02	(+)	**100%**	3.00E+05	(+)	33%	2.29E+02	(+)	**100%**
9	**3.56E+02**	**100%**	3.00E+05	(+)	80%	6.09E+02	(+)	**100%**	3.00E+05	(+)	67%	5.61E+02	(+)	**100%**
10	**3.24E+02**	**100%**	3.00E+05	(+)	40%	4.05E+02	(+)	**100%**	3.00E+05	(+)	34%	3.46E+02	(=)	**100%**
11	**2.40E+02**	**100%**	3.00E+05	(+)	50%	3.28E+02	(+)	**100%**	3.00E+05	(+)	50%	2.64E+02	(+)	**100%**
12	**1.43E+03**	**100%**	3.00E+05	(+)	20%	2.52E+03	(+)	**100%**	3.00E+05	(+)	20%	2.22E+03	(+)	**100%**
+/=/–			11/1/0			11/1/0			11/1/0			10/2/0		

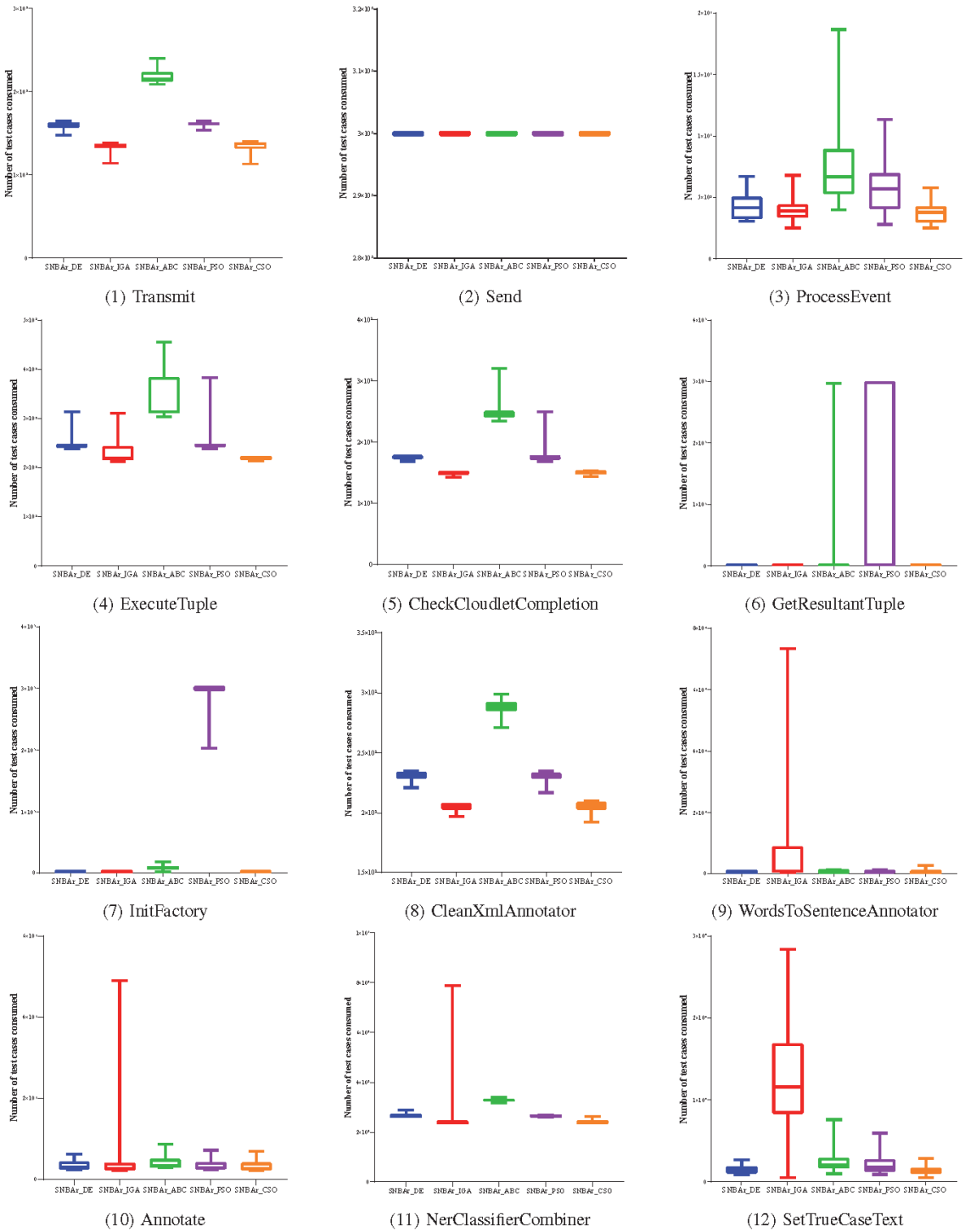

Figure 3 Box plots of the consumption of generated test cases between SNBAr-CSO with other search-based algorithms and their variants

From the above results, we can conclude that SNBAr can improve the ability of search-based algorithms to solve ATCG-PC problems by adding local search phase. Compared with SNBAr-DE, SNBAr-IGA, SNBAr-ABC and SNBAr-PSO, SNBAr-CSO is an effective method to solve ATCG-PC problems.

5.4 Comparison of Different Selection Strategies

Since schema, NBAr and scatter search are used in SNBAr-CSO, in order to discuss the influence of different strategies on the algorithm, we have carried out a strategy separation experiment, and the results are shown in Tab. 9.

In the comparison strategy contained in Tab. 9, scatter search is adopted by default when it is not specified. For example, CSO means CSO that only uses scatter search, S-CSO $_{ss\_}$ means CSO that includes schema but does not use scatter search, S-CSO means both schema and scatter search, SNBAr-CSO $_{ss\_}$ table means CSO that includes schema and node branch archive but does not include scatter search, SNBAr-CSO is the method we proposed in this paper.

Table 9 Experimental results of different selection strategies

No.	SNBAr-CSO		CSO			S-CSO$_{ss-}$			S-CSO			SNBAr-CSO$_{ss-}$		
	Ave.T	*Ave.c*	*Ave.T*		*Ave.c*	*Ave.T*		*Ave.c*	*Ave.T*		*Ave.c*	*Ave.T*		*Ave.c*
1	**1.34E+02**	**100%**	1.36E+02	(=)	**100%**	1.58E+03	(+)	**100%**	1.36E+02	(+)	**100%**	1.60E+03	(+)	**100%**
2	**3.00E+05**	**67%**	**3.00E+05**	(=)	49%	**3.00E+05**	(=)	**67%**	**3.00E+05**	(=)	**67%**	**3.00E+05**	(=)	55%
3	**3.70E+02**	**100%**	1.05E+04	(+)	**100%**	4.04E+03	(+)	**100%**	5.86E+02	(+)	**100%**	1.89E+03	(+)	**100%**
4	**2.19E+02**	**100%**	3.51E+02	(+)	**100%**	4.57E+04	(+)	99%	3.24E+02	(+)	**100%**	1.03E+04	(+)	**100%**
5	**1.49E+02**	**100%**	1.53E+02	(+)	**100%**	4.83E+04	(+)	98%	1.50E+02	(=)	**100%**	5.24E+04	(+)	98%
6	**2.35E+02**	**100%**	8.64E+02	(+)	**100%**	1.87E+05	(+)	82%	9.87E+02	(+)	**100%**	2.39E+04	(+)	98%
7	**1.43E+03**	**100%**	6.98E+03	(+)	**100%**	2.61E+05	(+)	97%	6.22E+03	(+)	**100%**	1.16E+04	(+)	**100%**
8	**2.05E+02**	**100%**	**2.05E+02**	(=)	**100%**	2.05E+04	(+)	**100%**	**2.05E+02**	(=)	**100%**	4.37E+03	(+)	**100%**
9	**3.56E+02**	**100%**	2.51E+03	(+)	**100%**	1.01E+05	(+)	97%	8.56E+02	(+)	**100%**	9.91E+04	(+)	97%
10	**3.24E+02**	**100%**	4.37E+02	(+)	**100%**	1.70E+04	(+)	**100%**	4.11E+02	(=)	**100%**	5.46E+03	(+)	**100%**
11	**2.40E+02**	**100%**	5.67E+02	(+)	**100%**	4.99E+04	(+)	**100%**	5.39E+02	(+)	**100%**	4.86E+04	(+)	**100%**
12	1.43E+03	**100%**	**1.32E+03**	(=)	**100%**	3.00E+05	(+)	59%	1.89E+03	(+)	**100%**	2.23E+05	(+)	90%
+/=/–			8/4/0			11/1/0			8/4/0			11/1/0		

Based on the results in Tab. 9, we can see that SNBAr-CSO shows the best results among all the results. The combination of each strategy can improve the efficiency of the algorithm, and there is no conflict between the strategies. The scatter search strategy has a great influence on the algorithm, because compared with the quartile search, the scatter search can quickly find the best fitness individual in the search space of the target.

To sum up, different strategies used in SNBAr-CSO can be well combined, and each strategy solves different problems in the process of test case generation.

6 CONCLUSION

In this paper, we propose a competitive swarm optimizer with schema and node branch archive for the complex ATCG-PC problem with multiple variables in nodes. SNBAr-CSO combines the good exploration and search ability of CSO with schema and node branch archive (NBAr). A priori knowledge schema is helpful to quickly find out all variables affecting nodes. Node branch archive can record the relationship between nodes and variables. Schema and NBAr reduce a lot of search space when covering the target path, thus reducing the generation of a large number of redundant test cases. In order to prove the effectiveness of SNBAr-CSO, this paper studies its performance on 12 open source benchmark programs on two commonly used tool kits iFogSim and CoreNLP. It is compared with a variety of most advanced algorithms, some search-based algorithms and their combination with SNBAr variants. The experimental results prove the following conclusions. Firstly, the performance of SNBAr-CSO is better than GPE-IS, MISA, NBAr-DE and RP-DE. Second, SNBAr can well combine different search-based algorithms to solve ATCG-PC problems, and the combination of SNBAr and CSO is better than DE, IGA, ABC and PSO. Thirdly, there is no conflict between different strategies in SNBAr-CSO. They can be well integrated to improve the efficiency of generating test cases.

In the future work, we will combine SNBAr with better local search strategy to form more effective ATCG-PC solutions, and further explore more a priori knowledge in the field of ATCG-PC to enhance the solution ability of the algorithm.

Acknowledgements

This work was supported by the Guidance Programs of Science and Technology Funds of the Xiangyang city (2020ZD32) and the Major Research Development Program of Hubei Province (No. 2020BBB092) and Hubei Superior and Distinct Discipline Group of "New Energy Vehicle and Smart Transportation".

7 REFERENCES

[1] Gay, G., Staats, M., Whalen, M., & Heimdahl, M. P. (2015). The risks of coverage-directed test case generation. *IEEE Transactions on Software Engineering*, *41*(8), 803-819. https://doi.org/10.1109/TSE.2015.2421011

[2] Luo, L. (2001). *Software testing techniques*. Institute for software research international Carnegie Mellon University Pittsburgh, PA.

[3] Jaffari, A., Yoo, C. J., & Lee, J. (2020). Automatic test data generation using the activity diagram and search-based technique. *Applied Sciences*, *10*(10), 3397. https://doi.org/10.3390/app10103397

[4] Dai, X., Gong, W., & Gu, Q. (2021). Automated test case generation based on differential evolution with node branch archive. *Computers & Industrial Engineering*, *156*, 107290. https://doi.org/10.1016/j.cie.2021.107290

[5] Liu, F., Huang, H., Su, J., Semujju, S. D., Yang, Z., & Hao, Z. (2021). Manifold-Inspired Search-based Algorithm for Automated Test Case Generation. *IEEE Transactions on Emerging Topics in Computing*. https://doi.org/10.1109/TETC.2021.3070968

[6] Li, Z., Harman, M., & Hierons, R. M. (2007). Search algorithms for regression test case prioritization. *IEEE Transactions on software engineering*, *33*(4), 225-237. https://doi.org/10.1109/TSE.2007.38

[7] Zamli, K. Z., Alkazemi, B. Y., & Kendall, G. (2016). A tabu search hyper-heuristic strategy for t-way test suite generation. *Applied Soft Computing*, *44*, 57-74. https://doi.org/10.1016/j.asoc.2016.03.021

[8] Huang, J. C. (1978). Program instrumentation and software testing. *Computer*, *11*(4), 25-32. https://doi.org/10.1109/C-M.1978.218134

[9] Lakshminarayana, P. & Suresh Kumar, T. V. (2021). Automatic generation and optimization of test case using hybrid cuckoo search and bee colony algorithm. *Journal of Intelligent Systems*, *30*(1), 59-72. https://doi.org/10.1515/jisys-2019-0051

[10] Lijuan, W., Yue, Z., & Hongfeng, H. (2012, March). Genetic algorithms and its application in software test data generation. 2012 International Conference on Computer Science and Electronics Engineering, 2, 617-620. https://doi.org/10.1109/ICCSEE.2012.36

[11] Suresh, Y. & Rath, S. K. (2014). A genetic algorithm based approach for test data generation in basis path testing.

[12] Yao, X. & Gong, D. (2014). Genetic algorithm-based test data generation for multiple paths via individual sharing. *Computational intelligence and neuroscience*, 2014. https://doi.org/10.1155/2014/591294

[13] Lin, J. C. & Yeh, P. L. (2001). Automatic test data generation for path testing using GAs. *Information Sciences*, *131*(1-4), 47-64. https://doi.org/10.1016/S0020-0255(00)00093-1

[14] Huang, H., Liu, F., Zhuo, X., & Hao, Z. (2017). Differential evolution based on self-adaptive fitness function for automated test case generation. *IEEE Computational Intelligence Magazine*, *12*(2), 46-55. https://doi.org/10.1109/MCI.2017.2670462

[15] Sahoo, R. R. & Ray, M. (2020). Pso-based test case generation: A fitness function based on value combined branch distance. *Advanced Computing and Intelligent Engineering*, 589-598. https://doi.org/10.1007/978-981-15-1483-8_49

[16] Huang, H., Liu, F., Yang, Z., & Hao, Z. (2018). Automated test case generation based on differential evolution with relationship matrix for IFOGSIM toolkit. *IEEE Transactions on Industrial Informatics*, *14*(11), 5005-5016. https://doi.org/10.1109/TII.2018.2856881

[17] Gong, D., Tian, T., Wang, J., Du, Y., & Li, Z. (2020). A novel method of grouping target paths for parallel programs. *Parallel Computing*, *97*, 102665. https://doi.org/10.1016/j.parco.2020.102665

[18] Cheng, R. & Jin, Y. (2014). A competitive swarm optimizer for large scale optimization. *IEEE transactions on cybernetics*, *45*(2), 191-204. https://doi.org/10.1109/TCYB.2014.2322602

[19] Saadatjoo, M. A. & Babamir, S. M. (2019). Test-data generation directed by program path coverage through imperialist competitive algorithm. *Science of Computer Programming*, *184*, 102304. https://doi.org/10.1016/j.scico.2019.102304

[20] Fraser, G. & Arcuri, A. (2012). Whole test suite generation. *IEEE Transactions on Software Engineering*, *39*(2), 276-291. https://doi.org/10.1109/TSE.2012.14

[21] Mala, D. J., Mohan, V., & Kamalapriya, M. (2010). Automated software test optimisation framework - an artificial bee colony optimisation-based approach. *IET software*, *4*(5), 334-348. https://doi.org/10.1049/iet-sen.2009.0079

[22] Saboncchi, A. K. S. & Akay, B. (2020). Cryptanalysis of polyalphabetic cipher using differential evolution algorithm. *Tehnički vjesnik*, *27*(4), 1101-1107. https://doi.org/10.17559/TV-20190314095054

[23] Shang, R., Zhang, W., Li, F., Jiao, L., & Stolkin, R. (2019). Multi-objective artificial immune algorithm for fuzzy clustering based on multiple kernels. *Swarm and Evolutionary Computation*, *50*, 100485. https://doi.org/10.1016/j.swevo.2019.01.001

[24] Han, X., Wang, Y., Cai, C., Hou, X., & Wang, L. (2020). An Efficient Universal Bee Colony Optimization Algorithm. *Tehnički vjesnik*, *27*(1), 320-332. https://doi.org/10.17559/TV-20180516081110

[25] Sengupta, S., Basak, S., & Peters, R. A. (2019). Particle Swarm Optimization: A survey of historical and recent developments with hybridization perspectives. *Machine Learning and Knowledge Extraction*, *1*(1), 157-191. https://doi.org/10.3390/make1010010

[26] Liu, F., Huang, H., Yang, Z., Hao, Z., & Wang, J. (2019). Search-based algorithm with scatter search strategy for automated test case generation of NLP toolkit. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *5*(3), 491-503. https://doi.org/10.1109/TETCI.2019.2914280

[27] Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., & Buyya, R. (2017). iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience*, *47*(9), 1275-1296. https://doi.org/10.1002/spe.2509

[28] Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., & McClosky, D. (2014, June). The Stanford CoreNLP natural language processing toolkit. *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 55-60. https://doi.org/10.3115/v1/P14-5010

[29] Cai, G., Su, Q., & Hu, Z. (2021). Automated test case generation for path coverage by using grey prediction evolution algorithm with improved scatter search strategy. *Engineering Applications of Artificial Intelligence*, *106*, 104454. https://doi.org/10.1016/j.engappai.2021.104454

[30] Bouchachia, A. (2007, September). An immune genetic algorithm for software test data generation. 7th International Conference on Hybrid Intelligent Systems (HIS 2007), 84-89. https://doi.org/10.1109/HIS.2007.37

[31] Steel, R. G. & Torrie, J. H. (1980). *Principles and procedures of statistics: a biometrical approach*. New York, McGraw-Hill.

Contact information:

Xiaohu DAI, Postgraduate Student
School of Computer Engineering, Hubei University of Arts and Science,
No. 296, Longzhong Road, Xiangyang, Hubei, 441053, China
E-mail: xiaohudai@cug.edu.cn

Bin NING, Professor
(Corresponding author)
School of Computer Engineering, Hubei University of Arts and Science,
No. 296, Longzhong Road, Xiangyang, Hubei, 441053, China
E-mail: ningbin2000@hbuas.edu.cn

Qiong GU, Professor
School of Computer Engineering, Hubei University of Arts and Science,
No. 296, Longzhong Road, Xiangyang, Hubei, 441053, China
E-mail: qionggu@hbuas.edu.cn

Chunyang HU, PhD
School of Computer Engineering, Hubei University of Arts and Science,
No. 296, Longzhong Road, Xiangyang, Hubei, 441053, China
E-mail: huchunyan@hbuas.edu.cn

Shuijia LI, PhD student
School of Computer Science,China University of Geosciences, Wuhan,
No. 388 Lumo Road, Wuhan, Hubei, 430074, China
E-mail: shuijiali@cug.edu.cn