Abstract—One of the key problems in path testing is building a path through specified set of state-mates particularly which contain loops. Traditional genetic algorithm has been successfully used in software testing activities such as finding test data, selecting test cases and test cases prioritization. In this paper, we introduce a new variable length genetic algorithm. Based on the new algorithm, we present a new strategy for automatically generating a set of basis test paths which can be used as testing paths in any basis path testing technique. We define all elements of genetic algorithm such as chromosome representation, crossover, mutation, and fitness function to be compatible with path generation. In addition, we present a case study to show the efficiency of our strategy.

Keywords—Genetic Algorithm; Basis Path Testing; Path Generation.

I. INTRODUCTION

Structural testing requires the execution of a set of test paths in the program under testing. Generating this set of paths is a critical task, and its automation is strongly desirable for easing the testing process. This task can influence the efficacy and cost of testing activity.

Basis path testing is one of the very powerful structural testing criteria. Basis path testing requires number of test paths (basis set) equals to the cyclomatic complexity of program [1] such that every path is an independent path and all edges in the control-flow graph (CFG) are covered by all paths in the basis set. In addition, a path that is not contained in the basis set can be constructed by a linear combination of paths in this set.

Some path generation methods have been introduced so far. Bertolino and Marre [2] provided a path generation method by using a reduced CFG. Although all the statements and branches can be covered by the set of paths, it cannot be assured that the set of paths generated by this method is a basis set of paths. Pool [3] discussed a basis set of paths generation method on the depth-first search in CFG. It uses a recursive search in the CFG. As Pool’s method, the loop is not taken into account. In addition, this method did not consider how to choose the successor of multiple-successors node in a CFG to build a basis path during the construction of the basis set of paths. Guangmei et al. [4] presented an automatic generation method of basis set of paths which is built by searching the CFG by depth-first searching method. In order to avoid that the algorithm will never stop, and for reducing the searching procedure, the sub-path from the multi-indegree nodes to the end node of a program and the sub-path that contains a loop are recorded during the construction of a basis path. This method did not handle infeasible paths. Yan and Zhang [5] presented a method for generating a finite set F of feasible paths which satisfies the basis path coverage criterion. Then, they found a minimal subset S of set F such that S satisfies the test coverage criterion. The first step should check the feasibility of all paths and feasibility checking is quite time-consuming. Zhonglin and Lingxia [6] and Qingfeng and Xiao [7] use cyclomatic complexity in generating a set of linearly independent paths. Many basis paths are infeasible because of data dependences exist in variables involved in decision node. They combine the baseline method with the dependence relationship to avoid selecting infeasible paths. These methods didn’t handle loops.

Search-based testing techniques especially traditional genetic algorithms have been successfully used in software testing activities such as finding test data [8, 9]. Bint and Site discussed a path generation method based on genetic algorithm [10]. The defect of this method is that the generated set of paths cannot cover all edges of the CFG because the loop operation is removed. In addition, this method cannot generate a basis set of test paths.

In this paper, we introduce a new variable length genetic algorithm. Based on the proposed algorithm, we introduce a new technique for automatically generating a set of basis test paths. We define all elements of the new genetic algorithm such as representation, crossover, mutation, and fitness function to be compatible with path generation process. We will introduce how the proposed technique can handle the loop and infeasible paths. In addition, we will present a case study to show the efficiency of our new strategy.

The rest of the paper is organized as follows. Section 2 introduces the problem formulation. Section 3 gives some definitions. Section 4 presents our proposed strategy. Section 5 introduces a case study of the proposed method. Section 6 concludes the paper.

II. BASIS PATH TESTING

Structural testing generally requires the execution of a set Q of paths in the program under testing. Determining Q is a very important and critical task, and its automation is strongly
An independent path is a path of a graph that satisfies the following three conditions:

1. Every path should be an independent path.
2. All edges in a CFG should be covered by all paths in the basis set.
3. Every path that does not contain in a basis set of paths can be constructed by the linear operation among paths in this set.

The path contained in a basis set is called a basis path. The problem of this work is defining a new genetic algorithm and using it for generating a set Q of basis paths.

### III. BASIC CONCEPTS

We introduce here some basic concepts that will be used through this work.

#### A. Genetic Algorithms Principles

The basic concepts of genetic algorithms (GAs) were developed by Holland [12]. The GAs start by creating an initial population of individuals, each represented by randomly generated binary string called chromosome. The basic algorithm of GAs, where P(t) is the population strings at generation number t, is as follows:

1. Initialize P(t);
2. Evaluate P(t);
3. while termination condition not satisfied do
4. select P(t+1) from P(t);
5. recombine P(t+1);
6. evaluate P(t+1);
7. t = t + 1;
8. end while

In the evaluation step, the fitness of each individual is determined. The selection step is used to find pairs of individuals that will be combined in some way to contribute to the next generation. The process of crossover involves two chromosomes swapping chunks of data. Mutation introduces slight changes into a small proportion of population and is representative of an evolutionary step. The above algorithm will iterate until the population has evolved to form a solution to the problem, or until a termination condition is satisfied.

#### B. Program representation

A program’s structure is analyzed on the program flowgraph, i.e., an annotated directed graph which represents graphically the information needed to select the test cases.

A control-flow graph (CFG) is a directed graph $G=(N,E)$, with two distinguished nodes — a unique entry $(n_0)$ and a unique exit $(n_f)$. $N$ is a set of nodes, where each node represents a statement, and $E$ is a set of directed edges, where a directed edge $e=(n,m)$ is an ordered pair of adjacent nodes, called tail and head of $e$, respectively.

A dd-graph (DDG) is a digraph $G=(N,E)$, where $N$ is a set of nodes and $E$ is a set of edges, with two distinguished edges $e_0$, $e_1$ (the unique entry edge and the unique exit edge, respectively), such that any other edge in $E$ is reached by $e_0$ and reaches $e_1$, and such that for each node $n \in N$, $n \neq \text{tail}(e_0)$, $n \neq \text{head}(e_1)$, $(\text{indegree}(n) + \text{outdegree}(n)) > 2$, while $\text{indegree}(	ext{tail}(e_0)) = 0$ and $\text{outdegree}(	ext{tail}(e_0)) = 1$, $\text{indegree}(	ext{head}(e_1)) = 1$ and $\text{outdegree}(	ext{head}(e_1)) = 0$. An edge $e$ in a dd-graph DDG is an ordered pair of adjacent nodes, called tail and head of $e$, respectively (i.e., $e = (\text{tail}(e), \text{head}(e))$. A path $p$ of length $l$ in a dd-graph DDG is a sequence $p = e_0, e_1, e_2, \ldots , e_l$, where $\text{tail}(e_{i+1}) = \text{head}(e_i)$ for $i = 1, 2, \ldots , l-1$. A path $p$ is simple if all its nodes, except possibly the first and last, are distinct. A complete path in a dd-graph DDG is a path from the entry node to the exit node of DDG. Given a path $p = e_0, e_1, e_2, \ldots , e_l$, then a path $p' = e_0, e_1, \ldots , e_j$ from $e_i$ to $e_j$, with $1 \leq i \leq j \leq l$, is called a subpath of $p$. Antonia Bertolino and Martina Marre provided a procedure to construct the dd-graph by reducing the control flow graph of the program [2]. Figure 1(a) gives an example program, Figure 1(b) shows its control-flow graph, and Figure 1(c) provides its ddgraph.

In our proposed system, we will represent the program under test as a dd-graph according to the above definitions.

#### IV. OUR PROPOSED STRATEGY

#### A. Our new genetic algorithm

In this section, we present our proposed GA for automatic generation of basis test paths for the tested software, which uses a new fitness function to evaluate the generated test path. This fitness function depends on the concepts of the number of the adjacent edges in the ddgraph of the software under test.

The algorithm searches for test paths that satisfy the three conditions of the basis set of path (see section II). The major components of this GA are discussed below.
1) The search space of our algorithm:

The search space is the set of all solutions among which the desired solution resides. Each point in the search space represents one possible solution. Suppose that DDG is a ddgraph of a program under test. The length of the vector is the number of edges in the chromosome. In other words, the input domain of our algorithm is the set of decimal numbers. We randomly generate PS integer chromosomes to represent the edges in the ddgraph. An example the search space of the example program in Figure 1, all individuals have the form 0, 9 (\(e_9\), \(e_0\)). The minimum length of the chromosome is two and the maximum length equals the number of ddgraph’ edges.

4) Evaluation function

The algorithm can use any fitness function to evaluate the generated test path. The fitness function which depends on the concepts of the probability of the adjacent edges in the path is suitable for path generation. The algorithm uses this fitness function to evaluate each test path.

The fitness value \(f(v_i)\) for each chromosome \(c_i\) \((i = 1, ..., PS)\) is calculated as follows:

\[
 f(c_i) = \sum_{j=1}^{d} w_j \tag{1}
\]

where, \(d\) is the number of adjacent edges in the chromosome \(i\), \(w_j\) is the weight (probability) of the edge \(j\) in the chromosome \(i\), \(w_j = \frac{1}{L}\), where \(L\) the length of the chromosome (the number of edges in the chromosome).

The fitness value is the only feedback from the problem for the GA. A test case that is represented by the chromosome \(v_i\) is optimal if its fitness value \(f(v_i) = 1\).

Consider Figure 1(c), suppose 0, 1, 9 and 0, 2, 3, 9 are two chromosomes. The first chromosome contains three genes or edges \((L=3)\) while the second contains four genes \((L=4)\). The probability of each gene in the first chromosome is \(\frac{1}{3}\) and \(d=2\) while the probability of each gene in the second chromosome is \(\frac{1}{4}\) and \(d=3\). According to “(1)”, the fitness values of the first and second chromosomes are \(\frac{2}{3}\) and \(\frac{3}{4}\), respectively.

5) Selection

After computing the fitness of each test path in the current population, the algorithm selects test paths from all the members of the current population that will be parents of the new population. In the selection process, the GA uses the roulette wheel method [13]. This method is described below.

For the selection of a new population with respect to the probability distribution based on fitness values, a roulette wheel with slots sized according to fitness is used. Such roulette wheel is constructed as follows:

- Calculate the fitness value \(f(v_i)\) for each chromosome \(v_i\) \((i = 1, ..., PS)\).
- Find the total fitness of the population \(F = \sum_{i=1}^{PS} f(v_i)\).
- Calculate the relative fitness value \(rft(v_i) = \frac{f(v_i)}{F}\).

For example, suppose the program in Figure 1. Let us consider an example chromosome: 0, 1, 3, 5, 9. So, the given chromosome corresponds to the path \(p = \{e_0, e_1, e_3, e_5, e_9\}\).

3) Initial population

As mentioned above, each chromosome is represented by a vector of decimal numbers. We randomly generate PS integer vectors of length 2 to represent the initial population, where PS is the population size. The appropriate value of PS is experimentally determined. Each individual in the initial population contains two edges the entry and the exit. For the example in Figure 1, all individuals have the form 0, 9 \((e_9, e_0)\).
Calculate the cumulative fitness value $\text{cft}$ for each chromosome $\text{cft}(v_i) = \sum_{i=1}^{\text{ft}(v_i)} \text{ft}(v_i)$.

The selection process is based on spinning the roulette wheel $\text{PS}$ times; each time we select a single chromosome for a new population in the following way:

- Generate a random number $r$ from the range $[0..1]$.
- If $r < \text{cft}(v_i)$ then select the first chromosome $v_i$; otherwise select the $i$-th chromosome $v_i (2 \leq i \leq PS)$ such that $\text{cft}(v_i) \leq r < \text{cft}(v_{i+1})$.

Some chromosomes would be selected more than once.

6) Reproduction

In the Reproduction phase, we use three operators, crossover, mutation and breeding (a new operator we have proposed it), which are the key to the power of GAs. These operators create new individuals from the selected parents to form a new population.

Crossover: It operates at the individual level. During crossover, two parents (chromosomes) exchange sub-vector information (genetic material) at a random position in the chromosome to produce two new vectors (offspring). The objective here is to create better population over time by combining material from pairs of (fitter) members from the parent population. Crossover occurs according to a crossover probability. The probability of crossover $PX$ gives us the expected number $PX \times PS$ of chromosomes, which undergo the crossover operation. We proceed in the following way:

For each chromosome in the parent population:
- Generate a random number $r$ from the range $[0..1]$.
- If $r < PX$ then select given chromosome for crossover.

Now, we mate selected chromosomes randomly: For each pair of coupled chromosomes we generate a random integer number $pos$ from the range $[2..L-1]$ ($L$ is the number of edges in a chromosome). The number $pos$ indicates the position of the crossing point. Two chromosomes $(b_1, ..., b_{pos}, b_{pos+1}, ..., b_n)$ and $(c_1, ..., c_{pos}, c_{pos+1}, ..., c_n)$ are replaced by a pair of their offspring $(b_1, ..., b_{pos}, c_{pos+1}, ..., c_n)$ and $(c_1, ..., c_{pos}, b_{pos+1}, ..., b_n)$.

Suppose that $C_1 = (0, 1, 3, 5, 9)$ is a chromosomes and $pos = 3$. The edge at position 4 is $e_4$. The successors of $e_4$ are $e_{5}$ and $e_5$. We randomly select one of the successors and insert it at position 5. The new chromosome is $B_1 = (0, 1, 3, 4, 6, 9)$.

Mutation: It is performed on a cell-by-cell basis. Mutation always operates after the crossover operator, and changes each cell with the pre-determined probability. The probability of mutation $PM$, gives us the expected number of mutated cells $PM \times L \times PS$. Every cell (in all chromosomes in the whole population) has an equal chance to undergo mutation. So we proceed in the following way:

For each chromosome in the current population and for each cell within the chromosome:
- Generate a random number $r$ from the range $[0..1]$.
- If $r < PM$ then mutate the cell by replacing the edge with another edge of its siblings (edges with the same parent are called siblings such as $e_4$).

Suppose that $C_1 = (0, 1, 3, 5, 9)$ is a chromosomes. The second cell which has the value 1 will mutate to the value 2. Therefore the new chromosome will be $B_1 = (0, 2, 3, 5, 9)$.

Breeding: It is a new operator. We have developed this operator to enhance the chromosomes to get a complete path. It performed on a cell level. Breeding always operates after the mutation operator. We generate a random integer number $pos$ from the range $[2..L-1]$. The number $pos$ indicates the position of the breeding point. So, we proceed in the following way:

For each chromosome in the current population:
- Generate a random integer number $pos$ from the range $[2..L-1]$, $L$ is the length of the chromosome. We can put $pos = d$ (number of adjacent edges in the chromosome).
- Identify the edge at the position $pos$ and randomly select one edge of its successors.
- Then, insert the successor edge at the position $pos+1$ and increase the length of the chromosome by one.

Suppose that $C_1 = (0, 1, 3, 4, 9)$ is a chromosomes and $pos = 4$. The edge at position 4 is $e_4$. The successors of $e_4$ are $e_5$ and $e_5$. We randomly select one of the successors and insert it at position 5. The new chromosome is $B_1 = (0, 1, 3, 4, 6, 9)$.

7) Elitist

The elitist function enhances the current population by storing one copy of the best member of the previous population. If the best member of the current population is worse than the best member of the previous population it exchanges them, and the best member of the current population would replace the worst member of the current population. After that, it stores the best member of the current population.

8) The Stop Conditions

In the traditional GA approach the population would evolve until one individual from the whole set which represents the solution is found. In our case, this condition would correspond to finding groups of path achieving the basis test paths conditions. The evolution stops when a set of individuals has satisfied the required conditions and its fitness value $f(v) = 1$. The solution is this set.

The algorithm will stop and the search will end in two cases. The first case when the generated test paths satisfy the conditions of the basis set of path. The second case when the number of generation reaches the maximum number of generation.

B. The Overall Algorithm of the Strategy

Our proposed GA-based strategy accepts as input the program to be tested, the control-flow graph (CFG) and the ddgraph (DDG) of the program, the entry ($e_0$) and the exit ($e_n$) edges of the ddgraph (DDG) and the set $S_i$ of successors of each edge $e_i$. Also, it accepts the GA parameters: population size, maximum number of generations, and probabilities of the crossover and mutation operators. The algorithm produces a set of basis test paths.

The algorithm generates one basis test path at a time and repeats until the required paths are obtained or the maximum number of generations is exceeded. The overall algorithm is presented in Figure 2.
A GA algorithm to automatically generate set of basis test paths for a given program */

Input:
The program to be tested \( P \);
The control-flow graph (CFG) and the ddgraph (DDG) of \( P \).
The entry \( e_0 \) and the exit \( e_n \) of the ddgraph (DDG).
The set \( S \) of successors of each edge \( e_i \).
Population size (PS);
Maximum no. of generations (MG);
Probability of crossover (PX);
Probability of mutation (PM);
Output:
Set of basis test paths (BTP) for \( P \).

\[ w_i = \frac{1}{5} \text{ for } i = 1, ..., 5 \]

Therefore, the

\[ \sum_{i=1}^{5} w_i = 1 \]

Find the set of successors for each edge in the ddgraph.
Pass the control-flow graph, ddgraph, and table of successors to the test path generation module.

TABLE I

<table>
<thead>
<tr>
<th>edge</th>
<th>0_1</th>
<th>0_2</th>
<th>0_3</th>
<th>0_4</th>
<th>0_5</th>
<th>0_6</th>
<th>0_7</th>
<th>0_8</th>
<th>0_9</th>
<th>0_10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successors</td>
<td>e_1</td>
<td>e_2</td>
<td>e_3</td>
<td>e_4</td>
<td>e_5</td>
<td>e_6</td>
<td>e_7</td>
<td>e_8</td>
<td>e_9</td>
<td>e_10</td>
</tr>
</tbody>
</table>

\[
S_i \text{ successors of each edge } e_i
\]

In the following paragraphs, we give a brief description for our proposed module for path generation of C++ programs.
This module uses the new genetic algorithm to solve the problem of deriving a set of basis test paths.

2) Path Generation Module

The path generation module uses the suggested genetic algorithm to generate set of basis test paths. This module will start by initializing all paths by the entry and exit edges. Then it will increment each path by adding new edges until get a complete path. For example \( \{ e_0, e_2 \} \) is the initialization of any path then the algorithm will add new edges according to the operators of the genetic algorithm until get a complete path such as \( \{ e_0, e_2, e_4, e_5, e_6 \} \). The algorithm will repeat this process until generating the set of basis test paths or satisfying a stop condition.

V. A CASE STUDY

In this section, we introduce a case study to show how the proposed strategy can find a basis set of test paths for the example program in Figure 1. In the following, we will show all the steps of our proposed genetic algorithm. Suppose that the population size (PS) is 4.

1) Initial Population:
The initial population is four individuals each one contains the entry and the exit edges only. Table 2 shows the four chromosomes of the initial populations.

2) Evaluation of the Current Population

Suppose that the current population has the following four chromosomes: \( C_1=(0, 1, 3, 5, 9), C_2=(0, 2, 3, 4, 9), C_3=(0, 2, 3, 5, 9) \) and \( C_4=(0, 1, 3, 4, 9) \). To find the fitness value of each chromosome, we convert each chromosome into the corresponding path. Then, we use equation (1) to find the fitness value for the chromosome. In each chromosome there are five edges in its corresponding path \( (L=5) \), and four adjacent edges \( (d=4) \). Then, \( w_j = \frac{1}{5} \text{ for } j = 1, ..., 5 \). Therefore, the

Figure 3: The block diagram of the proposed tool

Figure 2: The Overall Algorithm.

C. A Basis Test Paths Generation Tool

We are implementing a basis test paths generation tool based on our proposed strategy. This tool consists of two main modules:

1) The Analysis Module.

2) Path Generation Module.

Figure 3 shows the overall diagram of our proposed tool. We give a more details of these two modules of our tool in the following subsections.

1) Analysis Module

The analysis module has been built to perform the following tasks:

- Read the program under test.
- Classify program statements and reformat them to facilitate the construction of the program CFG.

- Construct the control-flow graph of the reformatted version of the program (see Figure 1(b)).
- Construct the ddgraph by reducing the control flow graph using the REDUCE algorithm \[2\]. Figure 1(c) gives an example ddgraph after applying the REDUCE algorithm on the control flow graph.

We read the program under test and reformat program statements to generate control flow graph (CFG) and the ddgraph (DDG) of the given program. The entry \( e_0 \) and the exit \( e_n \) of the ddgraph (DDG). The set \( S \) of successors of each edge \( e_i \).
An efficient method to generate feasible paths for testing programs involves selecting the parents of the next population, calculating the fitness of the new population and applying genetic operators. The fitness of a chromosome is calculated as follows:

\[ F = f(C_1) + f(C_2) + f(C_3) + f(C_4) \]

where \( f(C) \) represents the fitness of chromosome \( C \). Table II shows the fitness values of the chromosomes of the initial population.

**Table II: The Chromosomes of the Initial Population.**

<table>
<thead>
<tr>
<th>Chromosome #</th>
<th>Chromosome</th>
<th>Corresponding Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>0, 2</td>
<td>( e_0, e_2, e_1 )</td>
</tr>
<tr>
<td>C2</td>
<td>0, 2</td>
<td>( e_0, e_2, e_1 )</td>
</tr>
<tr>
<td>C3</td>
<td>0, 9</td>
<td>( e_0, e_9, e_2 )</td>
</tr>
<tr>
<td>C4</td>
<td>0, 9</td>
<td>( e_0, e_9, e_2 )</td>
</tr>
</tbody>
</table>

Fitness value of \( C_1 = f(C_1) = w_1 + w_2 + w_3 + w_4 = \frac{4}{5} = 0.8 \). Table III shows the fitness values of the current population.

**Table III: The Fitness Values of the Current Population.**

<table>
<thead>
<tr>
<th>Chromosome #</th>
<th>Corresponding Path</th>
<th>Fitness Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>( e_0, e_1, e_2, e_3 )</td>
<td>0.80</td>
</tr>
<tr>
<td>C2</td>
<td>( e_0, e_1, e_2, e_3 )</td>
<td>0.80</td>
</tr>
<tr>
<td>C3</td>
<td>( e_0, e_1, e_2, e_3 )</td>
<td>0.80</td>
</tr>
<tr>
<td>C4</td>
<td>( e_0, e_1, e_2, e_3 )</td>
<td>0.80</td>
</tr>
</tbody>
</table>

3) Selection:

We use the roulette wheel method to select the parents of the next population. The total fitness \( F = f(C_1) + f(C_2) + f(C_3) + f(C_4) = 3.2 \). Table IV shows the computations of roulette wheel.

**Table IV: The Roulette Wheel.**

<table>
<thead>
<tr>
<th>C #</th>
<th>Fitness value</th>
<th>Relative Fitness</th>
<th>Cumulative Fitness</th>
<th>( r )</th>
<th>Parents</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>0.80</td>
<td>0.25</td>
<td>0.25</td>
<td>0.70</td>
<td>C2</td>
</tr>
<tr>
<td>C2</td>
<td>0.80</td>
<td>0.25</td>
<td>0.50</td>
<td>0.20</td>
<td>C1</td>
</tr>
<tr>
<td>C3</td>
<td>0.80</td>
<td>0.25</td>
<td>0.75</td>
<td>0.80</td>
<td>C3</td>
</tr>
<tr>
<td>C4</td>
<td>0.80</td>
<td>0.25</td>
<td>1.0</td>
<td>0.15</td>
<td>C1</td>
</tr>
</tbody>
</table>

4) Crossover:

Suppose that the probability of crossover \( PX = 0.80 \). Therefore, the expected number of chromosomes is 4, where \( PX \times PS = 0.8 \times 5 = 4 \).

**Table V: The Selected Parents for Crossover.**

<table>
<thead>
<tr>
<th>Parents</th>
<th>( R )</th>
<th>The selected parents</th>
<th>New individual ( pos = 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pa1=C2</td>
<td>0.65</td>
<td>( e_0, e_2, e_1, e_3 )</td>
<td>( e_0, e_2, e_1, e_3 )</td>
</tr>
<tr>
<td>Pa2=C1</td>
<td>0.82</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Pa3=C3</td>
<td>0.87</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Pa4=C1</td>
<td>0.40</td>
<td>( e_0, e_2, e_1, e_3 )</td>
<td>( e_0, e_2, e_1, e_3 )</td>
</tr>
</tbody>
</table>

5) Mutation:

Suppose that the probability of mutation \( PM = 0.15 \). Therefore, the expected number of mutated cells is 3, where \( PM \times L \times PS = 0.15 \times 5 \times 4 = 3 \).

**Table VI: The Mutation Operation.**

<table>
<thead>
<tr>
<th>Current population</th>
<th>( r )</th>
<th>New population</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e_0, e_2, e_1, e_3 )</td>
<td>0.5, 0.1, 0.1, 0.2, 0.1</td>
<td>( e_0, e_2, e_1, e_3 )</td>
</tr>
<tr>
<td>( e_0, e_2, e_1, e_3 )</td>
<td>0.1, 0.6, 0.2, 0.1, 0.1</td>
<td>( e_0, e_2, e_1, e_3 )</td>
</tr>
<tr>
<td>( e_0, e_2, e_1, e_3 )</td>
<td>0.1, 0.1, 0.4, 0.4, 0.1</td>
<td>( e_0, e_2, e_1, e_3 )</td>
</tr>
<tr>
<td>( e_0, e_2, e_1, e_3 )</td>
<td>0.1, 0.2, 0.1, 0.1, 0.1</td>
<td>( e_0, e_2, e_1, e_3 )</td>
</tr>
</tbody>
</table>

6) Breeding:

Suppose that the random number \( pos = \) number of adjacent edges in the chromosome = 4. Then, insert the successor edge at position 5. Table VII shows the results of applying the breeding operator. After computing the fitness of the new population, we get two independent paths \( p_1 = e_0, e_1, e_2, e_3 \) and \( p_2 = e_0, e_2, e_3, e_1, e_0 \). The algorithm will repeat the steps from 3 into 6 to get other two independent paths \( p_3 = e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8 \) and \( p_4 = e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8 \). We can see that \( p_1 \) is a sub-path of \( p_3 \) and \( p_4 \) as well. Where the cyclomatic complexity of the directed graph in Figure 1 C(DDG) = 10 – 8 + 1 = 3. Therefore, the set of basis test paths consists of the three paths \( p_2, p_1 \), and \( p_3 \).

**Table VII: The Breeding Operation.**

We introduced a new genetic algorithm based strategy for generating a set of basis test path which can be used as testing paths in any basis path testing technique instead of selecting these paths manually. We defined all key elements of genetic algorithm such as chromosome representation, crossover, mutation, and fitness function. In addition, we present a case study to show the efficiency of our new strategy. Our future work concerns on doing more experiments to measure the efficiency of our strategy and compare it with other work. We will study how the proposed strategy can handle the loop.

**REFERENCES**


