Finding the Critical Path with Loop Structure for a Basis Path Testing Using Genetic Algorithm

Jakkrit Kaewyotha and Wararat Songpan (Rungworawut)

Department of Computer Science, Faculty of Science, Khon Kaen University, Khon Kaen, Thailand
jakkrit_k@kkumail.com, wararat@kku.ac.th

Abstract. Path testing is strongest code coverage in white box testing techniques. The objective of path testing is design path generating of program under test. However, testing all paths does not mean that will find all defects in a program especially loop structure program problem. In this paper significantly achieved used genetic algorithm for automatic finding the critical path. The GA can analyze control flow graphs of program to finding defect path called critical path with loop structure. In addition, the approach designs automation of generating critical paths, which the algorithm can be adopted to find an optimal solution in five programs under test. The experimental results shown that can generate and recommend a set of critical defect path for five programs in different loop structure of program. Our proposed approach is effective to help developer to find critical paths which means the paths should be improved in a program previously.

Keywords: White Box, Path Testing, Genetic Algorithm.

1 Introduction

The software testing processing has spending time and high cost, according with the objective of testing the software in order to given the performance and accurate with the requirement. In the recently, the several software testing companies where interested for high accuracy and improved the testing quality to reduce errors in software also known as defects. The defect is weak point of the software's source code or the software function faults which brought the incorrect for the expectance output. The several importance factors for software testing both spend high cost and times. Many defects arise because programmers have forgotten to include some processing in their code, so there are no paths to execute. Some defects are also related to the order in which code segments are executed. Wherever, software testers have effort to decrease the times and cost for software testing to improve the performance and reduce in spending time. Therefore, the increasing of the automated software testing researches was growing which separated for two types: black box and white box techniques. Black box technique only is pointed the software input and output. In other hand, white box technique represented the testing for software structure which focuses on the software's source code. However, the problem is how to adopt this technique to

H. Unger et al. (eds.), *Recent Advances in Information and Communication Technology 2015,*
Advances in Intelligent Systems and Computing 361, DOI: 10.1007/978-3-319-19024-2_5

complex source code in order to real program can have both sequence structure and loop structure to implement source code completely. When source code is executed, it shows only error parameter within a line. Therefore, the basis path testing is used in this problem to analyze defects that found on whole paths. We propose an approach to generate the priority path of software which applied the genetic algorithm with loop structure that is the major contribute in this paper. In addition, we increased the parameter adjustment for find out the defect to solve loop structure program that is challenge. For example, population size, crossover, mutation and number of generation by using genetic algorithm. The paper is arranged as follows: Section 2 presents the related work. Section 3 gives definition of basis path testing and how to construct the control flow graph. Section 4 presents experiments an approach that uses genetic algorithm applied to path testing with case study. The experimental results and discussion are described in the section 5. Finally, the conclusion present in section 6.

2 Related Work

There are many research in software testing with genetic algorithm. Hermadi et al. [1] presented using genetic algorithm generate test cases for white box testing of software testing which test cases is built by path testing which this paper analyze to control parameters affect to performance of genetic algorithm for path testing. The rest of the paper shows population size very impact for path coverage, in order is allele range while number of generation and mutation rate is low impact in term of number of paths found. The paper showed only the experimental results with parameters adjustment. Mansour and Salame [2] presented generate test cases for execute specified paths in program by using two algorithms are simulated annealing algorithm and genetic algorithm. These algorithms are based on the formulation optimization of the path testing problem which includes integer and real value. This paper compare three algorithms are simulated annealing algorithm, genetic algorithm and Korel's algorithm with eight subject programs. The rest of paper shows simulated annealing algorithm and genetic algorithm are efficient than Korel's Algorithm in term number of executed path, simulated annealing algorithm tends to executed better than genetic algorithm in term number of executed path and genetic algorithm faster than simulated annealing algorithm. They focused on comparison of three algorithms with generated test cases for programs. However, the algorithm does not specify critical paths to suggest how to fix it. Srivastava and Kim [3] presented method for optimizing software testing efficiency by identifying the most critical path clusters in a program by developed variable length of genetic algorithm in order to optimize and select software path, moreover, they tried to use weight to control path generation. The problem is genetic algorithm generated many paths and redundant when occurred the same fitness values and also not likely cause of error. Nirpal and Kale [4] presented apply genetic algorithm to generate test cases to test selected path. The genetic algorithm is used to selected path a target and executes sequence of operator iteratively for test cases. This research experiment triangle classification program and used genetic algorithm to automatic generate test cases for path testing and compare with method

generate test case produced by random. The experimental result shows genetic algorithm according to test data more effectively and quality of test cases by genetic algorithm is higher than test cases by produced by random. Ghiduk [5] presented genetic algorithm for generating test path and presented new technique for automatically generating a set of basis test paths. In this research, the basic structure of the software analyzes and design in form of control-flow graph (CFG) then converts CFG in form of dd-graph (DDG) to begin process of genetic algorithm a new technique was presented. This research aims to generated basis paths of each program which research was presented 10 programs. The experimental result shows their technique that can find path around 80% of actual basis paths of all subject programs which can be used testing paths in various parts of the technique in path testing and shows genetic algorithm of this paper effective in test path generation. Ngo and Tan [6] presented based approach to infeasible path detection for dynamic test data generation which is a method finding for infeasible path which can be apply with other dynamic path-oriented test data generation technique to increase efficiency the many techniques to finding infeasible path. This paper used control flow graph is representative structure software. The example software of paper is compute_grade and presented test data generation using infeasible path detection algorithm. This research was experiment to compare Bueno approach using 4 java system in experiment are PMD, JMathLib, GFP, SOOT. The experimental result shows the proposed method is very effective in finding infeasible paths with average precision of 96.02% และ recall of 100% in all the cases. The most papers solved with programs do not focus on loop structure. In difference, this paper showed how to solved the program with collected loop structure that is our challenge work.

3 Basis Path Testing

3.1 Path Testing

The path testing meaning the white box testing for assisted the tester who design the test case which measure by the cyclometic complexity of program graph. In addition, the cyclometic complexity measure could define the set of data for performing flow graph called directed graph and consists the node which including statement and edges represent the flow of control. For example, node i and j where the directed graph link between from i to j after i node is executed and j is computed immediately. Path testing is an approach to testing where you ensure that every path through a program has been executed at least once. The program graph is written from programming language which is transform to a directed graph called control flow graph (CFG). CFG represented the structure of program similar with the flowchart without the condition. The node of each control flow represented directed control flow graph including either assignment nodes or decision nodes. The statement defined by the line of code which the execution related with the path from started node to sequence node, which linkage between nodes are executed. For example a reserve number program performed by C language. The line of code i will be translated to node i of CFG as show in Fig 1. Between node i and j called edge and represented by e_n where n is

consequence number execution. The nodes from 1 to 5 are sequence nodes and the edges also are executed from e_1 to e_5, next from node 6 to 8 repeat the loop with edge e_6, e_7, e_8 and exit loop in node 6 with e_9 and direct to node 9 and 10 with e_{10} respectively.

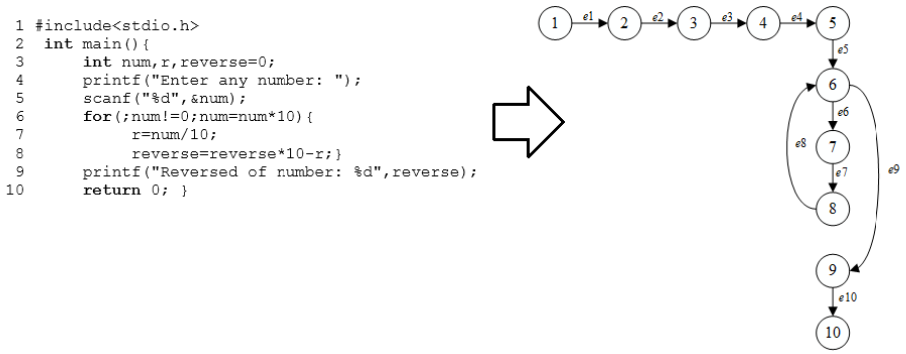

Fig. 1. Source code to CFG of Reserve number program

4 Experiments

In the process of finding the critical path could be compiled a genetic algorithm on a program under testing. In Fig 2 shows the applied genetic algorithm to path testing starts with the process of initial population generation which designs a set of many solutions which these population called chromosomes computed by fitness function. A critical path means the highest fitness function of those chromosomes. After completed fitness function computing then performs the selection process which the elitism method was selected. The elitism method is keeping the best chromosome according to fitness function. Then the Crossover state, random two chromosomes transform to parent and sharing a part of genes. The product of sharing stated called offspring and the mutation process was computed after completed the crossover process. Finally, swap some genes within an offspring. The algorithm can be repeated until the required number of generations is reached by population size. For example, using genetic algorithm find out critical path on reserve number program.

4.1 Initial Population

The number of population selected from all chromosomes which is random by the number of edge. The number of edge is randomized into each genes of a chromosome. The number of all edge depends on the program that translates to CFG. For example, reserve number program has 10 edges that are assigned randomly an integer only between 1 and 10 to each genes of a chromosome as show in Figure 3.

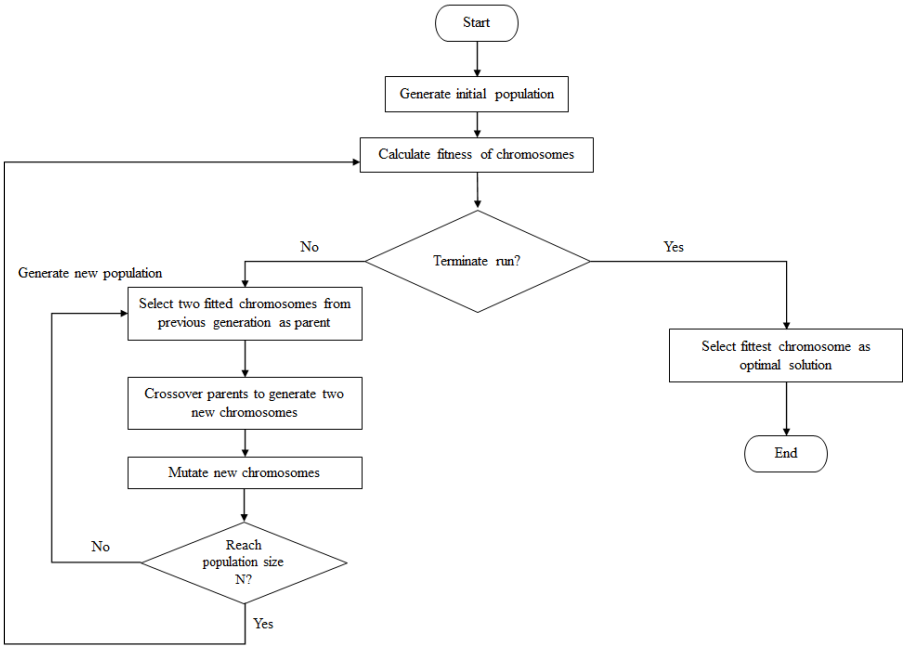

Fig. 2. Genetic algorithms process

Chromosomes 1	**10**	**5**	**6**	**7**	**8**	**1**	**2**	**9**	**4**	**3**
Chromosomes 2	**4**	**1**	**10**	**7**	**2**	**5**	**6**	**3**	**8**	**9**
Chromosomes 3	**2**	**7**	**4**	**1**	**10**	**3**	**8**	**9**	**6**	**5**

Fig. 3. Initial population generation

4.2 Fitness Function

The Fitness function is measure the fitness of chromosome of each chromosome where possible solution design. Each chromosome in population gives a fitness values. If the chromosome gives highest fitness value which means critical path. The fitness functions are represented follow as Eq.1

$$\text{Fitness Function} = \sum_{i=1}^{\#Edges} \text{Defect (i)}\,, \tag{1}$$

where edge i has linked with edge i+1

As Figure 1 reserve number program is drawn to CFG flow and stored defects into defect table (i.e., Fig. 4). The number of defect in each edge gets from the program executed on complier between lines of code that appeared error. And the table is collected defected into defect table. Therefore, a chromosome can be computed by fitness function in equation (1). For example, fitness value of chomosome1, chomosome2 and chomosome3 gives 3, 2 and 1 respectively. Let us look chomosome1 calculation; edge 10 no has linked with edge 5 from Fig. 4 that gives 0 from edge10 to edge 5, therefore, defect gives value 0. Next defect is calculated edge 5 that only has linked to edge 6; the fitness function is worked given 1. Therefore, the total of fitness value of chomosome1 gives 3.

Edge	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
Start 5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0
					End					

Fig. 4. Defect table of reserve number program

4.3 Crossover

Crossover process needs to choose two chromosomes by selection process as parents to sharing a part of genes and produce two offspring. For example, chomosome2 and 3 as parent crossover between them if position of crossover randomly is 5. The offspring1 is obtained from gene 1 to 5 of parent1 and from 6 to 10 of parent2. In addition, the offspring2 is obtained from 1 to 5 of parent2 and from 6 to 10 of parent1 in Fig. 5. However, for the algorithm will define a crossover rate. If the crossover is not greater than the rate, the parents will crossover process.

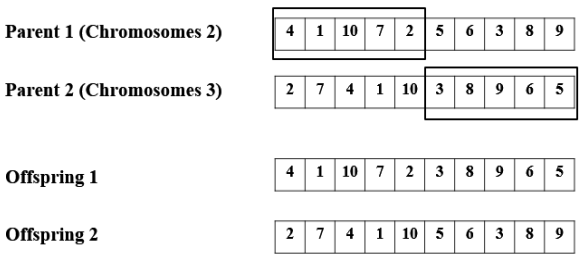

Fig. 5. Crossover process

4.4 Mutation

The mutation process will computed after the crossover process finished by swapping some genes within an offspring (e.g. position 2 and 8) as Fig. 6.

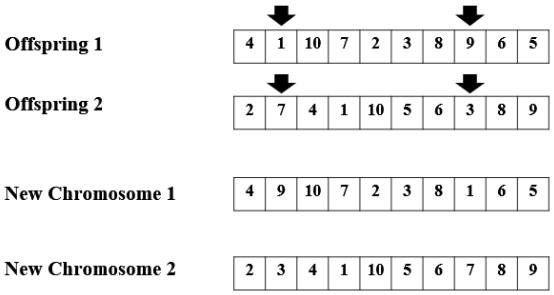

Fig. 6. Mutation process

5 Experimental Results

There are five programs for testing including as Reverse a number [7], Check prime number [8], Even number pyramid [9], Insert an element in an array [10] and Insertion sort [1,11,12], Which each program differences structure of program as show in Fig. 7-10. The objective of this experiment is to clarify the algorithm that discovers the defect path as critical path even more complex program of 5 programs.

```
1 #include<stdio.h>
2  int main(){
3       int num,i,count=0;
4       printf("Enter a number: ");
5       scanf("%d",&num);
6       for(i=2;i>=num/2;i++){
7            if(num%i!=0){
8             count--;
9                 break;}}
10     if(count==0 && num!= 1)
11          printf("%d is a prime number",num);
12     else printf("%d is not a prime number",num);
13     return 0;}
```

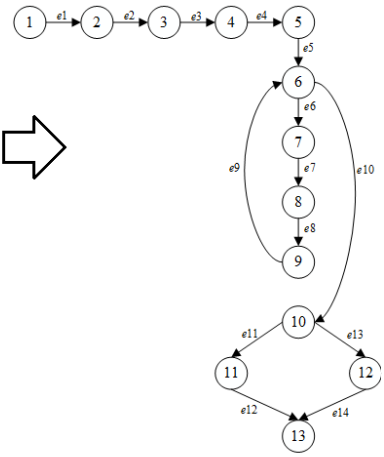

Fig. 7. CFG of Check prime number

```
1  #include<stdio.h>
2   int main() {
3      int i, j, num = 2;
4      for (i = 0; i < 4; i++) {
5         num = 2;
6         for (j = 0; j == i; j++) {
7            printf("%f\t", num);
8            num = num + 1;    }
9         printf("\n"); }
10     return (0);}
```

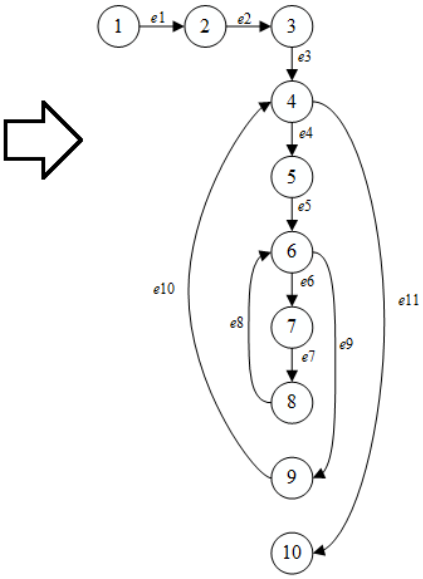

Fig. 8. CFG of Even number pyramid

```
1  #include<stdio.h>
2   int main() {
3      int arr[30], element, num, i, location;
4      printf("\nEnter no of elements :");
5      scanf("%d", &num);
6      for (i = 0; i < num; i++) {
7         scanf("%d", &arr[i]);    }
8      printf("\nEnter the element to be inserted :");
9      scanf("%d", &element);
10     printf("\nEnter the location");
11     scanf("%d", &location);
12     for (i = num; i != location; i--) {
13        arr[i] = arr[i + 1];    }
14     num++;
15     arr[location - 1] = element;
16     for (i = 0; i < num; i++)
17        printf(" %d", arr[i]);
18     return (0);}
```

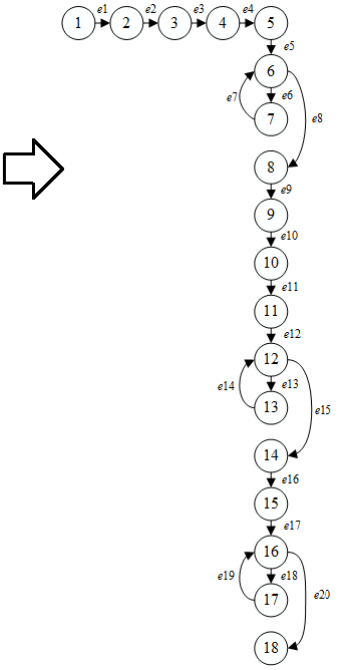

Fig. 9. CFG of Insert an element in an array

```
1 #include<stdio.h>
2  int main(){
3    int i,j,s,temp,a[20];
4    printf("Enter total elements: ");
5    scanf("%d",&s);
6    printf("Enter %d elements: ",s);
7    for(i=0;i<s;i++)
8        scanf("%d",&a[i]);
9    for(i=1;i<s;i++){
10       temp=a[i];
11       j=i-1;
12       while((temp<a[j])&&(j>=1)){
13       a[j-1]=a[j];
14           j=j+1; }
15       a[j+1]=temp;  }
16   printf("After sorting: ");
17   for(i=0;i<s;i++)
18       printf(" %d",a[i]);
19   return 0;}
```

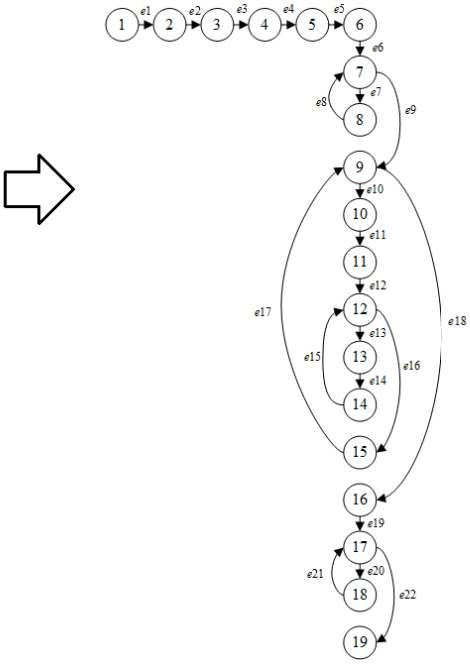

Fig. 10. CFG of Insertion sort

Start

Edge	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0

End

(a)

Start

Edge	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	0	1	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0

End

(b)

Fig. 11. Defect table of (a) Check prime number (b) Even number pyramid (c) Insert an element in an array (d) Insertion sort

Edge	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Start 10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

End

(c)

Edge	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Start 11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

End

(d)

Fig. 11. (*continued*)

We set up optimal set of parameters for algorithm running as bellows,

Table 1. Parameter setup

Programs	Population size	Crossover rate	Mutation rate	Number of generation
Reverse a number	2000	0.8	0.08	100
Check prime number	20000	0.8	0.08	100
Even number pyramid	2000	0.8	0.08	100
Insert an element in an array	2000	0.8	0.08	100
Insertion sort	20000	0.8	0.08	100

Table 2. Critical Paths of 5 programs in different structure

Programs	Path generation	Critical path	Fitness value
Reverse a number	*e*2-*e*4-*e*1- *e*3-*e*10- *e*5- *e*6- *e*7- *e*8-*e*9	*e*5- *e*6- *e*7- *e*8- *e*9	4
Check prime number	*e*4-*e*11- *e*14- *e*13- *e*12- *e*1- *e*5- *e*6- *e*7- *e*8- *e*9- *e*10- *e*3- *e*2	*e*5- *e*6- *e*7- *e*8- *e*9- *e*10	5
Even number pyramid	*e*3- *e*11- *e*2- *e*10- *e*1- *e*5- *e*6- *e*7- *e*8- *e*9- *e*4	*e*5- *e*6- *e*7- *e*8- *e*9	4
Insert an element in an array	*e*11- *e*3- *e*8- *e*17- *e*2- *e*1- *e*16- *e*12- *e*13- *e*14- *e*15- *e*20- *e*10- *e*18- *e*4- *e*5- *e*9- *e*7- *e*6- *e*19	*e*12- *e*13- *e*14- *e*15	3
Insertion sort	*e*6- *e*9- *e*10- *e*20- *e*2- *e*12- *e*13- *e*14- *e*15- *e*16- *e*4- *e*11- *e*19- *e*3- *e*8- *e*1- *e*17- *e*7- *e*22- *e*21- *e*18- *e*5	*e*12- *e*13- *e*14- *e*15- *e*16	4

In Table 2 presented the algorithm runs to solve finding critical path in five programs under test. For example, reverse a number program defect path is generated *e*2-*e*4-*e*1- *e*3- *e*10- *e*5-*e*6-*e*7-*e*8-*e*9, which expressed path defect *e*5- *e*6- *e*7- *e*8- *e*9 was generated represents the number of edge of program to analyze the critical path from reverse a number program. For example, number inputs 12345 for the reverse number program testing and expected result 54321 which difference with 29614540. Therefore, should consider solving this critical path in Fig. 12. The source code in line 6 is revised from num=num*10 to num=num/10, in line 7 is revised from r=num/10 to r=num%10 and in line 8 is revised from reverse*10-r to reverse*10+r

```
1 #include<stdio.h>
2  int main(){
3       int num,r,reverse=0;
4       printf("Enter any number: ");
5       scanf("%d",&num);
6       for(;num!=0;num=num/10){
7            r=num%10;
8            reverse=reverse*10+r;} }
9       printf("Reversed of number: %d",reverse);
10      return 0; }
```

Fig. 12. Example of revision reverse a number program by GA recommends

In addition, the programs of Check prime number, Even number pyramid, Insert an element in an array and Insertion sort program are generated to the critical path. The path *e*4-*e*11- *e*14- *e*13- *e*12- *e*1- *e*5- *e*6- *e*7- *e*8- *e*9- *e*10- *e*3- *e*2 of check prime number program gives the edge of defect in program that is *e*5- *e*6- *e*7- *e*8- *e*9- *e*10. The analysis of defect output when input 5 expected result is 5 is prime number but defect output 5 is not a prime number. However, developer will revise line 6-8 following path recommendation. The even number pyramid program is generated critical path as

*e*3- *e*11- *e*2- *e*10- *e*1- *e*5- *e*6- *e*7- *e*8- *e*9- *e*4 defect analysis is revised as line 6-8. The Insert an element in an array program is generated critical path as *e*11- *e*3- *e*8- *e*17- *e*2- *e*1- *e*16-*e*12- *e*13- *e*14- *e*15- *e*20- *e*10- *e*18- *e*4- *e*5- *e*9- *e*7- *e*6- *e*19 defect analysis is revised as line 12 and 13. And The Insertion sort program is generated critical path as *e*6- *e*9- *e*10- *e*20- *e*2- *e*12- *e*13- *e*14- *e*15- *e*16- *e*4- *e*11- *e*19- *e*3- *e*8- *e*1- *e*17- *e*7- *e*22- *e*21- *e*18- *e*5 defect analysis is revised as line 12-14.

6 Conclusion

The result of finding the critical path for software testing using genetic algorithm in that can supports the developer to analyze defects. Specifically, in this paper focused on the software including the loop structure which the most software was often found defect in loops structure. The newly found that successful how to apply genetic algorithm repository complex source code which adjustment of some parameter in the process as population size, crossover, mutation and number of generation. The fitness function is very robustness to loop structure due to the mainly finding critical path is to solve a continuous difficult finding critical path. Moreover, our approach helps to solve the path design problem in a complex program in different structure.

References

1. Hermadi, I., Lokan, C., Sarker, R.: Genetic Algorithm Based Path Testing: Challenges and Key Parameters. In: Proceeding of the 2010 Second World Congress on Software Engineering (WCSE), vol. 2, 241–244 (2010)
2. Mansour, N., Salame, M.: Data Generation for Path Testing. Software Quality Journal. 12(2), 121-136 (2004)
3. Srivastava, P.R., Kim, T.H.: Application of Genetic Algorithm in Software Testing. International Journal of Software Engineering and Its Applications, 3(4), 87-96 (2009)
4. Nirpal, P.B., Kale, K.V.: Comparison of Software Test Data for Automatic Path Coverage Using Genetic Algorithm. International Journal of Computer Science & Engineering Technology, 1(1), 12-16 (2010)
5. Ghiduk, A.S.: Automatic generation of basis test paths using variable length genetic algorithm. Information Processing Letters, 114(6), 304–16(2014)
6. Ngo, M.N., Tan, H.B.K.: Heuristics-based infeasible path detection for dynamic test data generation. Information and Software Technology, 50(7), 641–55 (2008)
Kumar, R., `http://www.cquestions.com/2008/01/c-program-to-reverse-any-number.html`
7. Kumar, R., `http://www.cquestions.com/2012/02/check-given-number-is-prime-number-or.html`
8. Taral P., `http://www.c4learn.com/c-programs/program-even-number-pyramid-in-c.html`
9. Taral P., `http://www.c4learn.com/c-programs/program-insert-element-in-array.html`
10. Kumar, R., `http://www.cquestions.com/2009/09/insertion-sort-algorithm.html`
11. Ahmed, M.A., Hermadi, I.: GA-based multiple paths test data generator. Computers & Operations Research, 35(10), 3107–3124 (2008)