Software Quality Control Based on Genetic Algorithm

S. Keshavarz and Reza Javidan, *Member, IACSIT*

***Abstract*—Software evaluation has a crucial role in the life cycle of software production system. Producing suitable data for testing the behavior of the software is a subject of many researches in software engineering. In this paper software quality control with criteria of covering application paths is considered and a new method based on genetic algorithm for generating optimal test data is proposed. In this algorithm, the fitness function, population production mechanism and other parameters of genetic algorithm is determined. In addition, the population production stopping criteria is based on critical edges of control flow graph. Critical edges are those that their presence in a control flow graph path represents the presence of other edges and the edges test shows the adequacy of graph test paths. The simulation results on prototype test data show the effectiveness of the proposed method.**

***Index Terms*—Software engineering, genetic algorithm, path covering, data.**

I. Introduction

Software architecture assessment includes evaluating of architectural decisions attributes and combination possibility of these attributes to access the expected quality features. Software quality control by producing suitable test data is a topic of interest in software engineering. One of the classical methods for software evaluation is white box test in which the structure of the program is used for testing the program. In this method, first the program structure is presented by control flow graph and then according to the graph, the program test is done. Control flow graph is a graph program which each node represents a program instruction and every edge represents a control transfer between two instructions of program.

White box test is divided into four types [1] based on the type of coverage provides for control flow graph: 1) white box test with instruction coverage criterion, 2) white box test with edge coverage criterion, 3) white box test with condition coverage criterion and 4) white box test with path coverage criterion. In the first method, the test must guarantee to run any instructions (passing over of each graph node) at least once. In the second method, test must guarantee to traverse every graph edge at least once. In the third method, test must guarantee the implementation of each condition of program (traversing of each Graph branch) at least once. In the fourth method, test must guarantee traversing of every program's (graph) path at least once. A path is a set of edges from the beginning of graph to its end. Since just a conditional instructions and function calling cause to separate paths in the program, a sequential non-conditional instruction is displayed with a node and the edge is not created between them.

The fourth method is the most complete kind of white box test but it faces with "many paths and being impractical test of all the paths" in complex (most paths) programs. A program with *n* branches (condition) has *2n* paths in maximum state that between them, some dependent paths could be. A dependent path is the path that each edge of it exists in previous traversal of graph paths at least so far.

In fact, if we traverse a program control flow graph using first depth and traversal path, this path is a dependent path. According to theory of McCabe [2], the control flow graph is a program with *n+ 1* independent path which *n* is the number of graph's branches. Therefore, independent paths of a program are an adequacy criterion test for all program paths.

Fig. 1 shows the control flow graph for program *P* that the sequential conditional instructions are displayed with a node and each condition is displayed with a separate node. If we traverse the graph with first depth method, six paths *P1* to *P6* is obtained that four paths, *P1*, *P2*, *P3* and *P5,* are independent. The *P4* and *P6* paths are repeated because their edges appeared in four previous paths.

An ideal data test set is a necessary and sufficient set. It means that its implementation by the program all paths causes to traverse all independent paths (necessary condition) and no additional path (dependent) is not traversed (sufficient condition). Producing such set has always been challenging so far and the different methods are presented for that.

Genetic Algorithm (GA) is one of the methods that can be used for generating optimal test data. In general, the idea of using genetic algorithms is be attention in software testing by researchers seriously [5, 8].

In this paper we suggest the genetic algorithm to produce this complex data. We will offer a fitness function that will be considered one of the following three adequate criteria of population production (set of test data): 1) all independent paths are traversed, 2) percent of the independent paths are traversed and continuing the producing of data does not increase the percent, or 3) the production time of the test data is not more than determined time.

The remainder of this paper is organized as follows: in Section II the white box test is explained. Genetic algorithm is briefly explained in Section III. The proposed software evaluation method based on generating optimal test data is the subject if Section IV. Experimental results and discussion is explained in Section V. Finally in Section VI conclusion and remarks are outlined.

Manuscript received March 16, 2011; revised July 20, 2011.

S. Keshavarz is with the Computer Engineering Department at Islamic Azad University, Arak branch, Arak, Iran (Corresponding author, e-mail: skc1359@gmail.com).

R. Javidan is with Department of Computer Engineering and Information Technology, Shiraz University of Technology, Shiraz, Iran (e-mail: reza.javidan@gmail.com).

```
1-if (x < > 0)            T={ s1: x=0; z=1; y=1;
2-   then if (y>1)            s2: x=1; z=2; y=1;
3-      then y =5;            s3: x=1; z=3; y=3;}
4-      else y = y – x;
5- else z=x;
6-if (z>1 && y > 1)
7- then z = z /x;
8-end
```

Program *p* and data sample for its testing

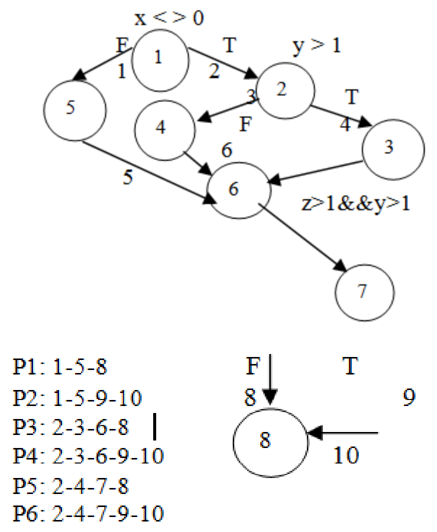

Fig. 1. Control flow graph of program P and its paths.

II. White Box Test

White-box testing is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are required and used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. While white-box testing can be applied at the unit, integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements

White box test or structural test is a test that tests the behavior of a program on the structure. Structure of a program is displayed by control flow graph (Fig. 1) and, as we said in the first section, it traverses with four methods. The fourth method is the most complete traversal which we describe it with an example.

Suppose for the program P test, we have produced T set test that includes three samples tests. Test sample s1 causes to implement 1, 5 and 6 instructions, test sample s2 causes to implement 1, 2, 4 and 6 instructions. Test sample s3 causes to implement 1, 2, 3, 6 and 7 instructions. Since the T test set has caused at least one to run per instructions, this set has adequacy of program test with covered criteria of instructions. Now, if we consider this set for traversal paths of program control flow graph, it is determined that s1,s2 cause to traverse independent path P1, P3 in order and s3 causes to traverse dependent path P6. Since this set does not cover some of the independent paths, it does not have the adequacy criteria for program. Now if we consider the test sample s4: x = 0; z = 2; y = 2, we see that program failed in 7th instruction. Thus, despite of the complete coverage of program instructions, T test set was not able to find the error. It means it is inadequate.

The idea comes to mind that if the control flow graph (or paths) of a program and an initial test set is available, how we can use the genetic algorithm to create suitable population from test's data that covers all paths of control flow. According to McCabe theory, the number of control flow graph's paths is equal to n+1 (n is number of branches), the number of test data that is needed to cover all paths will be equal to the McCabe's number. For example, the number of test data for program P is equal to 3+1 that three is the number of branches in Fig. 2.

if(x)… $\xi = \begin{cases} 0 & True \\ K & False \end{cases}$

if(x = = y)… $\xi = \begin{cases} 0 \\ abs(x-y) \end{cases}$

$x = y$

$x \neq y$

if(x<>y)… $\xi = \begin{cases} 0 \end{cases}$

Fig. 2. Fitness functions for the conditional relationships [5]-[6]

III. Genetic Algorithms

In the evolutionary algorithm set, genetic algorithms are multi-purpose and powerful optimization tools which model the principles of the evolution. They are capable of offering ideal solutions even in the most complex research atmospheres.

In the 60's, Genetic algorithms was presented by John Holland, mainly are used on issues that deal with the search space [3]. These algorithms try to simulate the evolution of living things through, find the appropriate answer to a question of between the possible values space.

Evolutionary algorithms are implemented on a set of encoded solutions, these solutions are chosen based on quality and then it is used as a basis to provide new solutions (this is sometimes done by changing the available components). In the past, the research mechanism was not related to the kind and range of the components. In the other word, the combination and the alteration of the components were done without knowing what the appropriate solutions were. But using independent operators have been documented to yield good results

Genetic algorithm is a method for the finite and infinite optimization problem-solving based on natural selection; it is a method which invokes the biological evolution. Population genetic algorithm changes the individual solutions repeatedly. In each phase, genetic algorithm selects some individuals by the selected method (like random method) out of the present population who are the parents and who are used for giving birth to the future generation. During the successive generations the population evolves into an optimized solution. Genetic algorithm is used to solve several optimization problems: it fits standard optimization algorithm and it includes problems in which the fitness function is discontinuous, stochastic or nonlinear. Genetic algorithm uses 3 major rules to make the next generation out of the present population in each phase.

Selection rules: select some individuals as parents who are involved in the next generation population.

Crossover rules: combine the parents for the creation of the next generation.

Mutation rules: uses the random alterations for the individuals' parents to give birth to the children.

In Initial level, a population (usually arbitrary) is selected from values (chromosomes) and then a new generation is created by doing the intercourse and their mutant. Individuals (chromosomes) of new population are evaluated by a function called the evaluation fitness function and Individuals with high fitness (powerful people) are selected to create the next generation. The genetic algorithm is a Directed random optimization method that moves towards the optimal solutions gradually. The idea of using these algorithms has been considered in software testing seriously [5]-[8].

IV. Proposed method

In this part software quality control with criteria of covering application paths is considered and a new method based on genetic algorithm for generating optimal test data is proposed.

In testing a program with the covered path criterion (you see parts 1 and 2), a test data is good data if cause to an independent traversal of a path. A path is a continuous edges set from start node to end node of program's control flow graph and an independent path is a path that at least one edge of it is not traversed by other paths. In software testing, the main worry is the automatic and ordered generation of data is necessary and sufficient for testing. Data is a necessary and sufficient only if it causes a traversal on an independent path.

In this section, we offer a systematic and automated procedure to generate data necessary and sufficient test of a program based on program control flow graph (see Section 2) and the covered aim of critical edges. We say the edge is a critical if its presence in a path represents an independent path (see Section 4-1). Thus, despite the critical edges in a path, we don't need to search for other edges in path, and it make close the search work of exponential order (n2) to linear order (n) because each edge represents a independent critical path and the number of independent paths have a linear order.

Steps of the proposed method is the following: (1) finding the critical edges of control the flow graph (genetic algorithm input), (2) determine the fitness function, (3) determine the covered table of critical edges, (4) determine initial data (chromosomes) (5) calculation of covered edges in graph with covered tables, (6) produce the generation and the fitted amount of each chromosomes, (7) determining corrupt chromosomes and replace them with optimal chromosomes, (8) restore output value and covered table. Steps 5 to 8 Repeat that the stopping criterion is according to user or consumed time. The user's opinion is asked based on results of Stages 5 and 8, which are declared to user. In Next, we will describe the steps of above method.

A. Determining the critical edges

For finding the critical edges of control flow graph, first, we divided the graph into classes that every class includes a simple branching (branch does not include any other branch) or a composite branch (nesting branches). Then for each class, we do the following operations: (1) – we select the innermost branch and mark it as critical edges, then we mark them and keep them in the critical edges set, (2) we mark other sub-path edges that includes the critical edge. This sub-path is a path that starts from the first edge class and ends with last node of that class. (3) If the unmarked edge remained in class, we return to first step. For make Clear in finding critical edges procedure, we go to work this procedure for Fig. 1 : the first step, the graph is divided into two class, the first class includes 1-6th nodes or two nested branches and the second class contains 6 – 8th nodes or a simple branch. In step 1,we consider the first class and find the innermost of its branch (Node 2), and we record its critical edges, the 3 and 4th edges, in CE = {3,4}, and mark them as viewed edges. In step 2, sub-paths are found that they include critical edges including, 2-3-6 and 2-4-7 sub-paths and its non-critical edges, 2, 6 and 7th edges, are marked. Now, 1 and 5th edges remain in the first class that we repeat the first step of algorithm that determines the rest of the critical edges. We identify the First Edge as a critical edge and add it into CE set and mark it as a viewed edge. So critical edges set from the first class is a set with three elements, 1, 3 and 4. In next step, edge 5 is marked as a next noncritical edge. Because other noncritical edges did not remain in the first class, we choose second class. This class has a simple branch that its critical edges, i.e. edges 8 and 9 are added to the CE set and CE = {3, 4, 1, 8, 9} is resulted. Critical edges 8 and 9 are marked and in the next step, the noncritical edge 10 is marked.

With identifying the critical edges, there is no need to review other edges of graph and their sub-paths. Hereby, the space of a sub-path is centered in a small part of the graph. It is worth noting that because of nested branches that are used in large and complex programs in greater depth and variety, the using of critical edges reduces in number and time of paths reviews noticeably.

B. Determine the tables of covered critical edges

As was expressed in section 4-1, the critical edges are represent the General Representative of control flow graph that with examining their mode (traversal or not traversal), we can estimate the percent of graph that covers test data. In fact, this estimate represents program instructions that are covered by this test data. Now we form a table called the covered table for critical edges and their corresponding instructions. In a similar way, [4] used the covered branches table that is much longer than the covered critical edges table that is used in this paper. Using this table, a stopping criterion is created for reproduction in genetic algorithms, that this criterion offers a reasonable number of traversal paths by the program which is close to the McCabe number. At the beginning, Covered critical edges table takes random data or user data that we call these data the candidate chromosome or the initial chromosome of genetic algorithms.

C. Fitness function

Since the fitness function in a genetic algorithm, plays an essential role and in the proposed method, this function plays a major role in directing the production of optimal test data, we introduce a fitness function and assess related conditions to critical edges with it: 1- control flow graph Formation of program and corresponding branches determination with the critical edges, and then finding the corresponding conditional instructions with their in the program, 2- Determining the critical edges, 3- Determining the logical propositions based on conditional instructions obtained from before step and making logical sentences with the conjunction and seasonal composition of propositions. Seasonal composition between propositions is for composite instruction such as IF and CASE that simultaneous implementation of them is not possible with a sample data. So if THEN and ELSE parts of a IF were in the corresponding instructions with a critical edges of graph, we use a combination of these parts to make a logical sentence and we use it for the remaining combination. 4 - determine the fitness function: the logical sentences obtained in step 3: (1) Each seasonal operator replace with the minimum and turning operator with the maximum function (2) Any conditional relationship is replaced with the equivalent function in Fig. 2. The result function will be the favorite fitness function. Fig. 2 functions are used as rules to build fitness functions [6]. In these functions (for example, fifth function in Fig. 2), because subtracting x and y is negative per true of condition, the function returns the zero value and otherwise (failure to establish the condition) value of subtracting two variables (the amount is positive).Of course like a second function, the return value is not clear that the value difference is positive or negative in not establishing of condition, we consider the absolute magnitude of it; you assume in the fourth function, x == y, then the function gain the x-y value that is zero that would be equal with the value of establish state in the same condition, so very small amount K (such as 10^{-6}) is added to it then the function restore different value conditions to establish modes.

D. Genetic Algorithms of proposed method

Steps 4 – 8 in the proposed method includes genetic algorithm that its pseudo code is expressed in Fig. 3. In this algorithm, we form the appropriate fitness function based on two functions in Fig. 2.

```
GC ( )
{
    Input:  Program: Changes version of program to be tested;
            InitData: Set of test data; CE: Crucial Edges;
    Output: Final: A solution test case set;
            CovTable: recorded CEs with status;
    #defines MaxTimes m; //Max acceptable time;
    #defines MC acceptable number of McCabe number;
    Variables declaration:
            CanCH1&2: Candidate chromosomes;
    TCE:Traversed Crucial l Edge;
    CovTable: Coverage Table;
    NextPop, CurPop: a set of test data;
    OpCH1&2: Optimal Chromosomes;
    Counter: iteration;
      Begin
    Step1: Make CanCH 1&2 by InitData;
           Get fitnessFUN() to Initial OpCH1 and CovTable;
           Initial CurPop;
    Step2: While (! fill CovTable with Y || counter <
           MaxTimes ||! Reach MC ||! User request) {
    Use Crossover and Mutation operations;
             Compute fitnessFUN ();
    Compute NextPop and Save OpCH2;
    Step3: for each chromosome of NextPop
             If (IsDefect (NextPop))
         Replace with OpCH1 one;
           CurPop = NextPop; OpCH1 = OpCH2;
    Step4: if (counter mod 10 == 0) {
             Compute number of TCE by CovTable;
             Show CovTable and Ask to continue ;} Counter++;
           } Final = CurPop;
           Return Final and CovTable;
       End.
    }
    Boolean IsDefect (chromosome)
    {  v = Fitness value of best OPCH1;
         if fitness value is less than v/l return true;
            else return false; //l is an optional value
    }
```

Fig. 3. Genetic Algorithm of proposed method

In the first step, data is produced by the user or randomly, the candidate chromosomes and equivalent binary of each it forms genes of chromosomes.

Then we run the program with the initial chromosomes (in fitness function) and for every critical edge is traversed (TCE in pseudo code), the value "Y" is inserted in the covered critical edges table for it. Second step of algorithm shows beginning of loop of data population production (chromosomes). In this loop, mating and mutation functions are called for each produced data, after that the fitness function determines the data fitness using of traveled edges. Thus the new generation is produced, whatever the amount of fitness will be more, and the chromosome is stronger and will have a more chance to participate in the next round. Since there is possibility that the optimal chromosomes in the current generation will be not product in the next rounds, we save them in the OPCH2 variable. If the fitness value of chromosome is less than a specified minimum amount, the chromosome corrupts (bad) and it will be removed from the present population (third step) and for compensate the shortage of chromosome in current population , we replace the most optimal chromosomes in the previous generation.

At the end of the loop, after several run of generation producing, we compute the number of traversal critical edges intervals through the covered table, and we display this number with covered table to the user (step 4), if the user want a request to continue the implementation of algorithm, the loop will be continued. We use a Counter variable for counting the number of repeated of population production, till it will not be more than certain quantity.

E. Stopping criterion of Genetic Algorithms

The proposed Genetic algorithm has four stopping criterion: (1) traversal of all critical edges (this traversal is studied with examining the covered table after each update), (2) proximity of the number of covered paths to the McCabe number, (3) the time limitations (this is based on the amount that the algorithm used it and if it is more that this amount, its performance loses) and (4) user satisfaction (after several run of the algorithm, covered table and number of covered paths are announced to user until user satisfaction that the algorithm will be stopped).

V. Plan a Problem

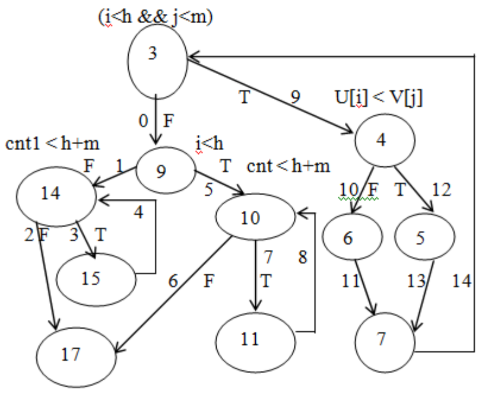

Fig. 4. Control flow graph of program Q

In this section, first we discuss a integration program of two arrays sorted (Program Q) that each array element is two elements, and then with applying the steps of proposed method, we produce the test data for these programs. In this program, U and V and S respectively the first, second and embedded arrays and h and m were presented sizes of the first and second arrays. fprintf instructions (instructions without number), are not the main instructions for the program Q and they have been added to this program for printing the numbers of graph control flow graph edges. Program control flow graph on program Q is shown in Fig. 4.

```
#1 void merge (int h,int m,const int*U,const int*V,int*S){
#2 int i = 0, j = 0, k = 0;
#3    while( i < h && j < m )    {
      fprintf(p,"9-");ph[9]++;
#4       if(U[i] < V[j]){
      fprintf(p,"12-");ph[12]++;
#5       S[k] = U[i];  i++;
          fprintf(p,"13-");ph[13]++;}
     else{fprintf(p,"10-");ph[10]++;
#6                S[k] = V[j];j++;
          fprintf(p,"11-");ph[11]++;
#7    }k++;
   fprintf(p,"14-");ph[14]++;
#8    }fprintf(p,"0-");ph[0]++;
#9      if (i < h){ fprintf(p,"5-");ph[5]++;
#10              for(int cnt = k; cnt<h+m;cnt++){
        fprintf(p,"7-");ph[7]++;
#11                     S[cnt] = U[i]; i++;
        fprintf(p,"8-");ph[8]++;}
#12                fprintf(p,"6-"); ph[6]++;   }
#13  else{  fprintf(p,"1-");ph[1]++;
#14          for(int cnt1 = k; cnt1<h+m; cnt1++){
        fprintf(p,"3-");ph[3]++;
#15          S[cnt1] = V[j]; j++;
        fprintf(p,"4-");ph[4]++;}
                fprintf(p,"2-"); ph[2]++;}
#16   }
```

Program Q- merging of two sorted arrays

Procedure fitnessFUN()

{

Run the program; Get TCE to Update CovTable;

$$\text{Return (Max}\left(\begin{cases}0 & A<B\\(A-B)+K & othetwise\end{cases}\right.,$$

$$\text{Min}\left(\begin{array}{l}\begin{cases}0 & cnt\le h+m\\(cnt-(h+m))+k & otherwise\end{cases}\\\begin{cases}0 & cnt1\le h+m\\(cnt1-(h+m)+K) & otherwise\end{cases}\end{array}\right))$$

}

Fig. 5. The proposed fitness function for evaluating the tes t data of program Q

P0: 0-1-2, P1:0-1-3-4-2,	P2:0-5-6, P3:0-5-7-8-6,
P4: 9-10-11-14-0-1-2,	P5:9-10-11-14-0-1-3-4-2,
P6: 9-10-11-14-0-5-6,	P7:9-10-11-14-0-5-7-8-6,
P8:9-12-13-14-0-1-2,	P9:9-12-13-14-0-1-3-4-2,
P10:9-12-13-14-0-5-6,	P11:9-12-13-14-0-5-7-8-6
CE = {2, 3, 6, 7, 10, 12}	

Fig. 6. Paths of program Q and its critical edges (CE)

Now we introduce the steps of proposed method for program Q. First step: determine the critical edges of control flow graph (Fig. 4): This graph consists of two classes that the first class of the nodes 3 to 7 and the second class of the nodes are 3 to 17. CE={10,12,7,6,3,2}is the critical edges set that 10 and 12th nodes from first class and others from second class.(see part 4.1). After obtaining the critical edges, e must form the fitness function. So for that, we find conditions of problem that have relative with critical edges. These conditions are U[i]<V[j] ‹cnt<h+m and cnt1<h+m. Then we make the following logical proposition:

$$\xi=\{\quad p(x)=(cnt\le h+m\ \cup\ cnt1\le h+m)\cap(A<B)$$

For making easy, we show U[i] with A and V[j] with B. now. Based on part 4.3 description, we transform this logical proposition to fitness function using of Fig. 2 function (Fig. 5). The graph of program Q (Fig. 4) has 12 paths (Fig. 6) that P0 ‹P1 ‹P2 ‹P3 ‹P4 and P8 paths are independent. In proposed method, we survey only critical edges (CE in Fig. 6) that show a considerable reduction compared to the paths in all other methods. May be reminded in the worst case, the number of paths of a program’s control flow graph is 2n that n is the number of branches in program's control flow graph.

A. Implementation of genetic algorithm for program Q

Whereas data input in program Q must be two sorted arrays, we consider that user has entered two initial chromosomes which each them contains two input and sorted arrays: arr1 and arr2.

CH1:[arr1{-1, 0}, arr2{1, 7}],
CH2:[arr1{-5, -4}, arr2{-1, 0}]

If we run program Q for these data, the traversal edges in graph of program Q look like table 1. Table 2 shows the covered critical edges for table 1 after running the first round of program Q.

N or Y of the CE's first column in table 2 shows the covered critical edge or no covered with CH1. N or Y of the CE's second column shows respectively the covered critical edge or no covered with CH2. With going on the run of genetic algorithm and stopping it, 6 chromosomes are produced as following that cover all critical edges. Table 3 shows that these chromosomes traverse respectively P2 ‹P0 ‹ P1 ‹P3 ‹P8 and P4 paths.

CH1:arr1 {-7, -8}, arr2:{-7, -8}, CH2:arr1 {-8, 7},arr2{7,8}
CH3:arr1{11,22}, arr2{15, 39},CH4:arr1{5, 6}, arr2{5, 6}
CH5:arr1{-1, 3}, arr2: {8, 13},CH6 arr6{-11, 0}, arr2{0,8}

TABLE I. TRAVERSAL EDGES OF FIG. 4 GRAPH PER TWO INITIAL CHROMOSOMES

#Edge	0	1	2	3	4	5	6	7	8
CH1	1	1	1	2	2	0	0	0	0
CH2	1	1	1	2	2	0	0	0	0

TABLE II. COVERED OF CRITICAL EDGES IN FIRST ROUND OF RUNNING PROGRAM Q

#Edges	9	10	11	12	13	14
CH1	2	0	0	2	2	1
CH2	2	0	0	2	2	2

TABLE III. THE TRAVERSAL EDGES OF FIG. 4 GRAPH PER FINAL CHROMOSOMES

#	Line(Node)	Predicate	Branch(#CE)	CE (Y:teste N: untested)

1	#4	If(A < B)	True(12)
			False(10)
				Y Y
				N N

2	#10	If(cnt < h+m)	True(7)
			False(6)
				N N
				N N

3	#14	If(cnt1 < h+m)	True(3)
			False(2)
				Y Y
				Y Y

VI. CONCLUSION

In this paper, a new method based on genetic algorithm to generate data of white box test is presented. In the proposed method we determined the fitness function, chromosomes to produce the optimum population and stopping criterion for the production of population based on critical edges of control flow graph program. White box test itself is based on structure of the programs. Therefore we cannot use it for programs that their source code or structures are not available and we must use black box test [6].

Compared with the same technique [5]–[6], since our approach is based on the critical edges, table cover, so covered table will be smaller and simpler and order times of path searching is converted from exponential degree to linear degree. Also with keeping the optimized Chromosomes at each stage, we lead the algorithm to come near to goal (producing appropriate test samples) more quickly. In this approach, though the steps of progress, user is known its improvement and type (the traversal paths) by us.

REFERENCES

[1] C.Ghezzi, M.Jazayeri and D.Mandrioli. Fundamentals of Software Engineering, 2nd Edition, Prentice-Hall, 2003.

[2] A.H.Watson, T.J. McCabe. Structural Testing: A Testing Methodology Using the Cylomatic Complexity Metric, Computer Systems Laboratory, National Institute of Standards and Technology Gaithersburg, MD 20899-0001, 1996.

[3] S.N. Sivanandam, S. N. Deepa. Introduction to Genetic Algorithms, Springer, 2007.

[4] J.Miller, M.Reformat, H.Zhang. Automatic test data generation using genetic algorithm and program dependence graph, Journal of Information and Software Technology, Elsevier, 48(7), pp. 586-605, 2006.

[5] I.Hermadi, M.A. Ahmed. Genetic Algorithm Based Test Data Generator, In Proceedings of IEEE Congress on Evolutionary Computation, Vol. 1, pp. 85-91, 2003.

[6] M.A.Ahmed, I.Hermadi. GA-based multiple paths test data generator, Journal of Computers & Operations Research, Elsevier, 35(10), pp. 3107-3124, 2008.