Eccentric Test Data Generation for Path Testing
Using Genetic Algorithm

Punam Mishra¹, Bhabani Shankar Prasad Mishra²
¹ Lecturer, Dept. of Computer Science, KISD, Bhubaneswar
email- Punam_mishra16@rediffmail.com
² Lecturer, CSE Department, Campus-3, K.I.I.T University, Bhubaneswar
email-mishra.bsp@gmail.com

Abstract. Effective and efficient test data generation is one of the major challenging and time-consuming tasks within the software testing process. Researchers have proposed different methods to generate test data automatically; however, those methods suffer from different drawbacks. In this paper we present a genetic algorithm-based approach that tries to generate a test data that is expected to cover a given set of target paths. Our proposed fitness function is intended to achieve path coverage that incorporates path traversal techniques, neighborhood influence, weighting, and normalization. This integration improves the GA performance in terms of search space exploitation and exploration, and allows faster convergence.

Keywords: Test data, Software testing, Genetic algorithm, Path traversal

1. Introduction

Testing is a very tiring, tedious and time consuming process. It is treated as a parameter for software quality assurance and represents the final review of specification, design, and coding. It also ensures that the software runs correctly as per customer requirement. Software testing accounts for 50% of the total cost of software development [1]. This cost could be reduced if the process of testing is automated. For software testing, test data have to be generated and prove them as better than other test data. Therefore, a systematic testing system has to differentiate 1-2 good (suitable) test data from bad test (unsuitable) data, and so it should be able to detect good test data if they are generated. Nowadays testing tools can automatically generate test data which will satisfy certain criteria, such as branch testing, path testing, etc.

In path testing every possible path in the program is tested. A path is nothing but set of conditions through which input travels from the starting point to the exit point. However, path testing is generally considered as an infeasible task because a program with loop statements can have an infinite number of paths. The path testing problem is considered as NP-complete problem, various test data generation methods for path testing have been proposed in the literature [4].

In this paper a genetic algorithm based approach has been proposed. The method is a metric which calculates the difference between the selected path to be traversed and the actual path taken by the input. A genetic algorithm tries to find the test data which traverses a given selected path is as follows:-

2. Proposed Algorithm

Euclidean distance [11] is taken to quantify the distance between two paths of the control flow graph. It is actually a metric which calculates the differences between two given path ie the target path and the exercised path. If there are branches for a particular path, then the testing may fail because two different paths may contain some common edge in different sequence. Therefore extended mode of the Euclidean
distance is used. It is derived from Pythagoras theorem. Euclidean distance in two dimensional forms is given by:

Let P and Q be two points in Euclidean space

\[ P = (p_1, p_2, \ldots, p_n) \]

and

\[ Q = (q_1, q_2, \ldots, q_n) \]

then distance is calculated as:

\[ \sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\ldots+(p_n-q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i-q_i)^2} \]

Normalized Euclidean distance is used so that values will always lie between \([0,1]\). Normalized Euclidean distance = Euclidean distance/range [12]

Now we can define a fitness function \( \text{CLOSENESS} = 1 - \) (normalized Euclidean distance)

1.1 Cascaded branch testing:

Let \( G \) = control flow graph

\( P = \{ \text{path}_i \mid \text{path}_i \) is a complete path within \( G \} = \{ \text{path}_1, \text{path}_2, \text{path}_3, \ldots, \text{path}_z \} \)

\( N = \) number of complete path of \( p \).

Let

\[ S_1^1 = \{ g : g \text{ branch of path}_1 \} \]

\[ S_1^2 = \{ h : h \text{ is a cascaded branch of path}_1 \} \]

\[ S_1^n = \{ k : k \text{ is a } n\text{-tuple cascaded branch of path}_1 \} \]

Continuing like this……,

\( S_q^t = \{ r : r \text{ is a } t\text{-tuple cascaded branch of path}_q \} \)

First order distance:

\[ D_{i-j}^1 = \sqrt{(x_1-y_1)^2+(x_2-y_2)^2} \]

\[ N_{i-j}^1 = D_{i-j}^1 / \text{range} \]

Continuing like this……,

\[ D_{i-j}^m = N_{i-j}^m = D_{i-j}^m / \text{range} \]

\( M_{i-j}^m = m\text{th order closeness between } i \text{ and } j \)

\( M_{i-j}^m = 0 \text{ or } N_{i-j}^m = 1 \) if path \( i \) and path \( j \) have no common branches

When \( N_{i-j}^1 = 1, N_{i-j}^2 = 1, \ldots, N_{i-j}^n = 1 \)

or

\( M_{i-j}^1 = 0, M_{i-j}^2 = 0, \ldots, M_{i-j}^n = 1 \)

It results in worst test case generation

When \( N_{i-j}^1 = 0, N_{i-j}^2 = 0, \ldots, N_{i-j}^n = 0 \)

Or

\( M_{i-j}^1 = 1, M_{i-j}^2 = 1, \ldots, M_{i-j}^n = 1 \)

It results in best test case generation

CLOSENESS can be computed as:

\[ M_{i-j}^1 \ast w_1 + M_{i-j}^2 \ast w_2 + \ldots + M_{i-j}^n \ast w_n \]

Where \( w = \) weighing factor which are chosen with experience

If \( \text{CLOSENESS } i-k > \text{CLOSENESS } i-j \), then path \( k \) is much closer to target than path \( j \).
3. Basic steps of path testing

1. Construction of control flow graph
2. Target path selection
3. Test data generation from the genetic algorithm and execution [6].

Fig. 1: Basic steps of test data generation.

For the testing process, triangle classifier problem is taken as a problem. The C code for triangle classifier problem is as follows:

```c
#include<string.h>

char path[256];

int triangleA(unsigned int a, unsigned int b, unsigned int c)
{
    int triangle=0;
    if((a+b>c)&&(b+c>a)&&(c+a>b))
    {
        strcat(path,"u");
        if((a!=b)&&(b!=c)&&(c!=a))
        {
            strcat(path,"y");
            Triangle=1;
        }
        else
        {
            strcat(path,"v");
            if(((a==b)&&(b!=c))||((b==c)&&(c!=a))||(c==a)&&(a!=b))
            {
                strcat(path,"z");
                Triangle=2;
            }
            else
            {
                strcat(path,"x");
                Triangle=3;
            }
        }
    }
    else
    {
        strcat(path,"w");
    }

    return(triangle);
}
```

538
Control flow graph construction:

![Control flow graph](image)

Fig. 2: Control-flow graph.

4. Conclusion

From the above discussion about genetic algorithm, we can conclude that genetic algorithm is used to generate test data automatically. The greatest merit of using the genetic algorithm is its simplicity. The proposed fitness function CLOSENESS is very simple and can enhance the speed.

5. References

[4]. Bruno T.Abreu,Eliane Martins,Fabiano L.deSousa.Automatic test data generation for path testing using a new stochastic algorithmSupported by Brazilian CNPq council through Reasearch Grant 134107/20047