Journal of Basic and Applied Engineering Research
Print ISSN: 2350-0077; Online ISSN: 2350-0255; Volume 1, Number 10; October, 2014 pp. 65-79

http://www.krishisanskriti.org/jbaer.html

Survey of Fitness Function Improvement for Unstructured Programs

Vivek Vashishta

Northern India Engineering College, GGSIPU, Dwarka, New Delhi, India

Abstract: **Automated testing technique has decreased the testing effort required to generate test data. The Evolutionary Algorithm uses fitness function to find the best suite of test data and hence guide the search process. But, unstructured program creates problem in the execution of Evolutionary Algorithm. The goal of this paper is to review some of the verified technique to transform the program in the structured way for better search guidance. The paper involves Boolean, exit, flag and && containing unstructured program and show their transformation methods. This leads to decrease in testing time, cost and make process less error prone. In this paper the software metrics values of the transformed and untransformed programs is compared and final conclusion is made to verify the low testing effort required for transformed program.**

Keywords: **Transformation, Fitness Function, Evolutionary Algorithm.**

1. INTRODUCTION

Testing plays crucial game in software verification process. However, it consumes resources like time and cost. Generating test using test criteria means for a set of test input to be sufficient. There are two classes of test criteria [3]: white box which depends on the structure of code and black box which depends on functionality of code. The goal of tester is to find the test data for test criteria. But it is tedious, time consuming and error prone process when done manually. So, automatic test data generation tool become the topic of interest. But, unstructured program creates problem for all automated test data generation. The goal of this paper is to show some of the methods to decrease the effect of unstructured program on test data generation.

High structural coverage of software is providing by Evolutionary Algorithm for specific test data. The necessary condition of Evolutionary Algorithm is the effective fitness function which gives better guidance in finding the search result. Evolutionary Algorithm is used for generating test data since they are applicable to differentiate different types of problems [10]. Evolutionary Algorithm works on the basis Darwin biological theory of evaluation. The overview of Evolutionary Algorithm is given below.

Evolutionary Algorithm for Test Data Generation [10]

BEGIN
INITIALISE population with random candidate solutions;
EVALUVATE each candidate;
REPEAT UNTIL (TERMINATION CONDITION is satisfied) DO
1 SELECT parent;
2 RECOMBINE pairs of parents;
3 MUTATE the resulting offspring;
4 EVALUVATE new candidates;
5 SELECT individuals for the next generation;
END

The algorithm works on the basis of Darwin theory where all the initial test data is taken as population. The goal of algorithm is to find the best suit of test data to accomplish the task. Various processes have been applied to filter the test data like mutation, combination and then the most suitable generation is selected.

Unstructured program gives output in terms of 0 and 1, which gives no guidance for test data generation. Evolutionary Algorithm takes help of transformed fitness function for finding the desired test data for the search guidance. Fitness function used in Evolutionary Algorithm to optimize the solution and give better result for search guidance. Different fitness functions emerge for search guidance through which test goal persuade. For example if a program has branch condition x==y the fitness function is |x-y| [11]. In node oriented method the fulfillment of aim is independent of part executed in control flow graph .The fitness function consists of approximation level and branch distance [10]. Approximation level: this metric assess how close and individual was to reaching the target on the basis of its execution part through control structure [10]. For a program, the flow is shown by arrow and the node represents the action. In the control flow graph, critical branch is simply a branch which leads to the target being missed. The second component of fitness function is branch distance. It is computed when control flow diverged away from the target down a critical branch [10]. The Control Flow Graph [3] for a program

consists of directed graph 'G'. The vertices represent the statement and directional flow is represented by edges. It is used to find the approximation level and critical branch of a program.

This paper focuses on presenting the transformation of unstructured programs into structured way through earlier verified techniques so that it gives some search guidance for Evolutionary Algorithm. The paper is organized as follows: section 2 describes the short circuit condition, section 3 describes flag condition, section 4 describes exit condition, section 5 describes the Boolean variables condition and section 6 presents the conclusion of paper.

2. SHORT CIRCUIT CONDITION

The main goal of fitness function is to find test data that suits test criteria. The goal of fitness function is to increase the chance of finding the solution and to give better guidance of the search to optimize it. Modifying the branch condition can improve the fitness function for the search [4].

To keep side effect unchanged the fitness function takes account the executed part result only. In the code given below, the true value depends upon the true value of all sub atomic elements in the expression [4].

if(a==0 && b==0 && c==0 && e==0 && f==0)

Example 1: Short Circuit Condition having && operator

There is low probability of coming solution by which fit all condition in expression. When one atomic condition is executed the chance of next condition to be fit decreases so there is need of conversion of expression in the effective way. The author [4] has given a solution to change the statement in another way which preserves functionality and make fitness function guidance better.

If (a==0) {
If (b==0) {
If (c==0) {

Example 2: Transformation of Example 1 in the form of "if loop"

In this method the evaluation of inner if statement earlier will lead to improvement mentioned in example 1. Calculation of all condition then prior to first then if statement with this the search of fitness function in process to perform better than that of 1. This leads to more guidance for test data generation.

3. FLAG CONDITION

Search based technique has difficulties in the branch coverage of last line as it is dependent of flag value in the code. The code is transformed in such a way to remove the flag problem. Evolutionary algorithm is used to search the output domain of the test object for desired test data.

1: flag=false;
2: if (a==0) flag=true;
3: ... /*
4: if (c == 4) {
5: if (flag && b> c)
6: /* test aim */

The execution of test aim on line 6 is depending on the condition of 2. The original approach searches for :

"c==4" and "flag=true && b>c".

This is created by the control dependency of the if-statements of line 4 and 5. Because of the last condition containing a flag an Evolutionary Testing behaves like a random search. The author [5] used the new fitness function for the better search algorithm in the test data.

The author [5] used the use definition analysis where the test aim is split into two node, first is the analysis node 2 and second is the node 5 for test aim. The improvement searches for

"a==0" and "c==4" and "flag=true && b>c"

The first part is created by the control dependencies for the flag assignment of line 2 and the second part by the control dependencies for the test aim. This new fitness function directs the search to "a==0" in the first instance, automatically resulting in fulfilling the flag condition. This improves the guidance of the fitness function.

4. EXIT CONDITION

Test data generation is the interested research topic. But unstructured programs create problem for automated test data generation. An unstructured program is the one which contains goto, return, flag, exit statement. In Genetic Algorithm, the test criteria is transformed to fitness function which evaluates how close the input comes to execute the desired branch. But the presence of unstructured program does not lead to the effective guidance of Genetic Algorithm.

When program has exit statement, it has two way of terminating the program through normal or through exit way. However, the test data generation will not know which of two ways will likely to give the desired output. So, there is need of transferring the program in the way to have single termination way.

For a transformed program the 100% branch coverage criteria is followed when an unstructured program p is transformed to structured program p' such that any input gives 100% branch coverage for p' also provides for p [3].

Branch coverage is computed using the set of test input and program under consideration. For example, the following program shows the transformation of multi exit statement into single exit statement.

```
while P do
S
if P_ then
S1;
exit n;
S2;
else
S3;
exit n;
S4;
fi
S5;
```

Example 3: Program with multiple exit statement

This is branch-coverage equivalent to the following program that contains only one exit statement in its loop [3].

```
while P do
S
if P_ then
S1;
else
S3;
fi
exit n;
S5;
```

Example 4: program with single exit statement

The author [3] has given a set of rules for transforming the program having multiple exit statement to single or none exit statement using skip method. The program is transformed using depth function which gives the value of exit level in the code and a respective transformation is applied on it.

The transformed program should follow the functional preservation of original unstructured program.

5. BOOLEAN VARIABLES

Test data generation through Evolutionary Algorithm search strategy to evolve candidate solution is the key component of Evolutionary Algorithm. Fitness function should be able to identify the best candidate and give guidance for the random search. For a case having output in the form of Boolean where either true or false is output. The Evolutionary Algorithm becomes random search. The author [1] gives new fitness function for better solution of the search.

```
public boolean isClosed()
{
If(size>= maxsize)
return 0 ;
else
return 1;
}
```

Here the isClosed method gives either true or false output and hence the Evolutionary Algorithm search becomes the random search.

The author [1, 2] gives better function by showing the value by which the wrong value diverted from the original required value and hence the new code for function is:

***f*(isClosed()) = *f*(size >= maxSize)**
=0 if (maxSize - size)≤ 0
Else
maxSize – size

The above mentioned code will give the direction to Evolutionary Algorithm for finding the search guidance and performs better than the earlier one.

6. COMPARISON OF SOFTWARE METRICS AFTER TESTABILITY TRANSFORMATION

To accomplish the goal of studying the effect of transformation on software metrics, I took a set of programs having flag problem and transformed them into the flag free version through the renowned approach of Harman [6] and Wappler [10]. I computed the software metrics of the program before and after transformation and noted down the results. Table 1 shows the computed result of metrics for the transformed version of program where '↑' shows increase in the values of the metric and '↓' shows decrease in the value of metrics.

Table 1: Metrics comparison after program transformation

LOC	Statement coverage	Branch	Method /class	Statement /method	Maximum complexity	Maximum Depth
↑	↑	↓	↑	↑	↓	↑

The tool used for the metric calculation is the Source Monitor [13] which computes all the relevant metrics of an unstructured c/c++ program. The tool is compatible with any version of operating system and it is freely available.

Analyzing the result through the source monitor tool, I found that the complexity of program decreases in the transformed program. Khalid [12] explained the relation between the complexity and testability in the paper. The paper states that the complexity of a program is directly proportional to the testing efforts. The more complexity the more testing effort required and less testability of the program.

Seeing the result of Table 1 I come to the conclusion that the complexity of transformed program decreased and hence its testability increases. The above result validates the transformed version attribute of the program.

7. CONCLUSION

Unstructured program containing flag, Boolean, exit and && operator degrades the performance of Evolutionary Algorithm. In this paper I presented transformation method of these unstructured functions as given by earlier researchers. Different methods are applied to transform the different kind of unstructured program to improve the fitness function. Comparing the software metrics value of transformed and untransformed program I come to the conclusion that transformed programs need less testing efforts. The improved fitness function leads to better search and give more search guidance for the test data generation. The improved Automation search through improved fitness function gives fast and accurate search method which assists the software developer to test their data fast.

REFERENCES

[1] Cheon, Yoonsik, and Myoung Kim. "A specification- based fitness function for evolutionary testing of object-oriented programs." *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM, 2006.

[2] Cheon, Yoonsik, and Kim Myoung. "A fitness function for modular evolutionary testing of object-oriented programs." (2005).

[3] Hierons, Robert M., Mark Harman, and C. J. Fox. "Branch-coverage testability transformation for unstructured programs." *The Computer Journal* 48.4 (2005): 421-436.

[4] Baresel, André, Harmen Sthamer, and Michael Schmidt. "Fitness Function Design To Improve Evolutionary Structural Testing." *GECCO*. Vol. 2. 2002.

[5] Baresel, André, and Harmen Sthamer. "Evolutionary testing of flag conditions."*Genetic and Evolutionary Computation—GECCO 2003*. Springer Berlin Heidelberg, 2003.

[6] Harman, Mark, et al. "Testability transformation." *Software Engineering, IEEE Transactions on* 30.1 (2004): 3-16.

[7] Binkley, David W., Mark Harman, and Kiran Lakhotia. "FlagRemover: A testability transformation for transforming loop-assigned flags." *ACM Transactions on Software Engineering and Methodology (TOSEM)* 20.3 (2011): 12.

[8] Gong, Dunwei, and Xiangjuan Yao. "Testability transformation based on equivalence of target statements." *Neural Computing and Applications* 21.8 (2012): 1871-1882.

[9] Li, Yanchuan, and Gordon Fraser. "Bytecode testability transformation." *Search Based Software Engineering*. Springer Berlin Heidelberg, 2011. 237-251.

[10] Wappler, Stefan, Andre Baresel, and Joachim Wegener. "Improving evolutionary testing in the presence of function-assigned flags." *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007. TAICPART-MUTATION 2007*. IEEE, 2007.

[11] Korel, Bogdan. "Automated software test data generation." *Software Engineering, IEEE Transactions on* 16.8 (1990): 870-879.

[12] Khalid, Sadaf, Saima Zehra, and Fahim Arif. "Analysis of object oriented complexity and testability using object oriented design metrics." *Proceedings of the 2010 National Software Engineering Conference*. ACM, 2010.

[13] www. sourcemonitor.com.