Research on Searching Algorithm for Test Data Generation

Wan Lin Zhang Wei Ma Xueyan
(Software Engineering Department Armored Force Engineering Institute Beijing 100072)

Abstract: **Test data generation is a complicated problem in software testing area and its method and technique are not mature. Program executing method show some advantages for its dynamic feature. But the searching algorithm can not be satisfactory. The paper introduces some methods for improving searching algorithm based on the understanding of typical algorithm. The goal is improving its efficiency and practicability.**

Key words: **software testing test case branch function searching algorithm**

I. INTRODUCTION

Software testing is very labor-intensive and expensive; It accounts for approximately 50% of the cost of a software system development. If the testing process could be automated, the cost of developing software should be reduced significantly. Of the problems involved in testing software, one is of particular relevance here: the problem of developing test data. Manual generation of test cases becomes very time-consuming and impractical. Efforts have been made to automate this process.

There are two types of automated test data generation: function test data generation and structural test data generation. One view of structural testing is to search the domain of the program for suitable inputs which will force the control flow along a predefined path. The most popular path-oriented methods are the static approach such as symbolic evaluation and the dynamic approach such as program execution.

Symbolic evaluation involves executing a program using symbolic values of variables instead of actual values. Once a path is selected, symbolic evaluation is used to generate a path constraint, which consists of a set of equalities and inequalities on the program's input variables. This path constraint must be satisfied for the path to be traversed. A number of algorithms have been used for the inequality solution. If the constraints are linear, then the path condition leads to a linear programming(LP) problem. LP problem can be solved by Simplex Method or by Karmarkar's algorithm. Solving a non-linear system is more difficult, sometimes impossible. Symbolic evaluation is a promising approach; however, there are still several problems which require additional research even for linear constraints: 1) evaluation of loops, 2) procedure calls, 3) the problem of array element determination and point determination. 4) the problem of infeasible path. 4) is general for any method; however, the first three can be resolved by applying dynamic methods.

Another approach of test data generation, which referred to as a dynamic approach, is based on actual execution of a program under test. Test data are developed using actual values of input variables. When the program is executed on some input data, the program execution flow is monitored. The key of this method is searching algorithms, which directly influence the efficiency of test case generation. Random method, function minimization method and genetic algorithms have ever been used for it. But they are all impractical due to its low-effect and limitation.

II. BACKGROUND

Form 1970s, research on automatically structural test case generation has developed. Random method and function minimization method have been used. From 1990s , genetic algorithms attract the attention in the test case generation.

Function minimization method is a traditional searching algorithms. It is good at finding the local optimum. Iterative improving is applied, and a new point is selected around current point for every iterative. But in direction exploration, it only can deal with the condition of single predicate. It can not get high probability of successful searching under the diversity of path predicates.

Genetic algorithms are loosely based on ideas from genetics and Darwinian evolution by combining samples to achieve new and fitter individuals. Different from the other

algorithms, it search randomly in the whole field, and evaluate every sample according to some strategy. Specific operations are used to optimize the samples until the optimum is located.

From the point of theory, genetic algorithms show its unique advantage and high efficiency in the problems of big space, multi-peak, non-linear and so on. But a series of implement problems make the algorithms dis-optimistic. For example, the coding project of parameter, the selection and control of operations, adjusting strategy of fitness value and the magnitude of original test cases group. If these problems can not be resolved perfectly, genetic algorithms can not be ideal too.

III. RESEARCH ON SEARACHING ALGORITHM

A. Branch function definition

In the program executing method, the most important aspect of using scout algorithms is the ability to define a suitable fitness function, which is a numeric measure of how close each sample test case is to the goal. And it will directly influence the efficiency of test case searching.

The form of branch function is first applied by Korcl. He assumes that the branch predicates are simple relational expressions (inequalities and equalities). That is, all branch predicates are of the following form: $E_1(x)opE_2(x)$, where $E_1(x)$ and $E_2(x)$ are arithmetic expressions, and op is one of $\{<,\leq,>,\geq,=,\neq\}$. Each branch predicate $E_1(x)opE_2(x)$ can be transformed to the equivalent predicate of the form: F(x) rel 0. F is a real-valued function, referred to as a branch function, which is 1) positive (or zero) when a branch predicate is false or 2) negative (or zero) when the branch predicate is true. It is obvious that F is actually a function of program input x. F and rel are given in Table 1:

TABLE 1 Branch Function

Branch Predicate	Branch Function F	rel
$E_1(x)>E_2(x)$	$F(x)=E_2(x)-E_1(x)$	$<$
$E_1(x)\geq E_2(x)$	$F(x)=E_2(x)-E_1(x)$	$\leq$
$E_1(x)<E_2(x)$	$F(x)=E_1(x)-E_2(x)$	$<$
$E_1(x)\leq E_2(x)$	$F(x)=E_1(x)-E_2(x)$	$\leq$
$E_1(x)=E_2(x)$	$F(x)=abs(E_1(x)-E_2(x))$	$=$
$E_1(x)\neq E_2(x)$	$F(x)=-abs(E_1(x)-E_2(x))$	$\leq$

Under the testing, branch function can describe the coverage of the fact path against the selected path in the testing unit.

The branch function defined by Korcl considered only simple predicates, i.e., predicates with single relational operator. The definition of branch function must be extended to the case of compound predicates containing boolean operators. F is given in Table 2:

TABLE 2 Extend Branch Function

Branch Predicate	Branch Function F
$E_1(x)$ and $E_2(x)$	$F(x)=Max(F_1(E_1(x)),F_2(E_2(x)))$
$E_1(x)$ or $E_2(x)$	$F(x)=Min(F_1(E_1(x)),F_2(E_2(x)))$
not $(E_1(x))$	$F(x)=-F_1(E_1(x))$

The extended branch function can still reflect the degree of departure between fact path and testing path.

B. improvement on searching process

In practical searching process, some details maybe directly influence the efficiency of searching. This section presents some methods from different aspects. The goal of it is to advance the algorithms' efficiency and its practicality.

1) multi-exploratory search

Due to the limitation of the branch function, function minimization method can only deal with the simple predicates. The problem of compound predicates is answered by extended branch function. But if the algorithms does not been improved, it will not deal with the compound predicates perfectly.

Successful exploratory search must satisfy the condition:

① one searching process can bring good change of correlative variable;

② the change can be transferred.

For simple predicates, if the condition 1 be satisfied, condition 2 be satisfied too. But for compound predicates, it will be complex, condition 2 will be an important problem.

For example, assume that the branch function is X>0&&Y<0 (X、Y are integer variable); its branch function is F=max (f_1, f_2) (f_1=0-X, f_2=Y-0). If the original test case is (-2, 2), F= max (0- (-2), 2-0) =2. In old function minimization method, positive searching is firstly applied to X variable. The exploratory value is X=-1. We can know that the value has brought good change of X variable because f_1'

=1<2= f_1. But due to F′=2=F, the change does not been transferred. Right direction will be lost, and searching will be failing finally.

Multi-variable exploratory search is considered in this paper. Among most direction combination, at least one direction combination is suited. The goal of exploratory search is to find the suitable direction combination, which can satisfy condition 1 and condition 2.

Single variable direction is expressed by the simplest binary code {0, 1}, 1 is positive direction, 0 is negative direction, then the all direction string of n variables is 2^n. It is not practical that searching that direction string orderly. How can we search the suitable direction string rapidly?

This paper present a method that indexing is adopted after grouping the direction bits.

Assume that the direction bits expression is $a_1a_2...a_n$ (n is the number of variables), $a_i \in M$. For multi-compound predicate, we consider the logic 'and' expression, which can be 'true' only when the value of every simple predicate is 'true'. We group all direction bits string to make every group has the same character, that is bit string in one group can be get by logic shifting of variable sequence. When the suitable bit string is in some group, matching exploration can get the information that there are even sub-predicate is 'false'. That is, we can search the group as:

$Q=\{A|\ A'=(a_1a_2...a_n)$ is bit string of every group, match (A, A') is even$\}$

(match (A, A′) is the number of dis-matching bits comparing A and A′)

In exploratory search, firstly direction group will be determinated, and orderly searching in the group, finally finding the direction string of the variables against the current compound predicate.

2) dynamic search step

In function minimization method, fixed step is adopted in pattern search. It is difficult to control for different program testing. It may be bring some problems:

• Assume that we select a large value for pattern move so that the branch function decreases in the first step of pattern search. In this case the method fails since another exploratory search results in the same direction.

• If we select a small value of pattern move, the number of pattern moves may be huge. The cause is that the magnitude of the step for the pattern move is roughly proportional to the number of successful steps previously obtained.

Aiming at this problem, we select the value of pattern move by dynamic method. Dynamic move can get information according to program executing, so that probing search can be avoided which improving the efficiency. The writer gets the value of pattern steps dynamically according to the branch function of collide predicate in the betterment algorithms of test data generation. The algorithms decrease the searching time and increase the efficiency. The dynamic move can be determined as:

$$Step=k_0+k*Value$$

(Value is the branch function value of current predicate; parameter k is determined by the coefficient of predicate expression; k_0 is a small correctional value; when the branch function value is 0, and the path is still not matching, the k_0 worked.)

Program errors are classified as computation errors and domain errors. A computation error occurs when a specific input follows the correct path, but the result of the computation is wrong. Domain error occurs if the program path traverses a wrong path for a specific input. From experience, test cases at the boundaries are prone to find errors, especially for domain errors. The test cases located by function minimization method and genetic algorithms have big randomicity in its field. Even generating successfully, it is hardly effective to testing.

We can know from the dynamic step method above that the algorithms is always firstly finding the point of $F_i(x)=0$. If the searching succeed, the test case is exactly on the subdomain boundaries. If the searching does not succeed, the k_0 can worked to make the test case be round the subdomain boundaries. (for example, for the 'true' branch of the predicate X>0, the test case searched usually is 1; for the 'true' branch of the predicate $X\geqslant 0$, the test case searched usually is 0;) As a result, the algorithms may produce test sets of high quality in the sense that they have a better chance of revealing errors at the subdomain boundaries.

3) influence variable

The TESTGEN test data generation system developed by Ferguson and Korel has a limitation: the alternative variable method used for optimization is simple but

inefficient as it does not use objective function derivative information. However, an important contribution is the use of data flow analysis as an aid in determining which variables affect a given branch predicate. This leads to an improvement in the efficiency of the optimization search.

Function minimization method and genetic algorithms are all considering the whole input variables of testing program, which may be increase the expending and lead to some invalid searching steps. In fact, only part of the input variables are responsible for the current value of the branch function. Experiments, several international examples were chosen, were performed to investigate the performance of the test data generation for varying numbers of generated input variables. The result is shown in Fig. 1:

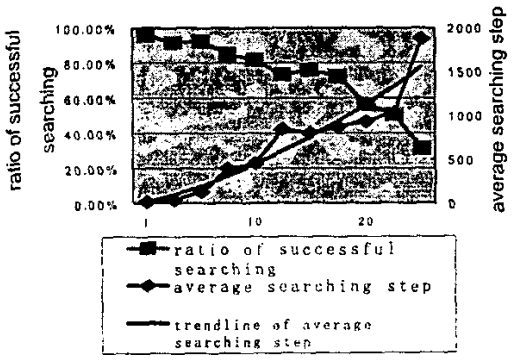

Fig.1 performance of the generation for varying numbers of variables

From the point of algorithms, the group mode we adopted in exploratory search makes a linear relation between the number of groups and the number of variables. And because the basis of every exploratory search is group, it is obviously that the times of executing is proportional to the number of groups. From Fig.1, the trendline shows that there is a close to linear relation in certain range between the average times of executing and the number of searching variables.(we can not exclude the other influence factors to the average times of executing, such as original test cases.) As a result, the little the dimension of searching variables, the little the times of searching, and the higher the success ratio of searching. Along with the increasing of the dimension of searching variables, the times of executing increase and the ratio of successful searching decrease. When get to certain limitation, the times of executing increase greatly and the ratio of successful searching decrease greatly, which shows that the algorithms is better to those predicates containing lesser variables. In fact, data flow analysis has important value, which can determine those input variables which are responsible for the current value of the branch function on the given program input and improve the searching efficiency.

The method of determining influence variable for current predicate is flexible. Data flow analysis techniques such as slicing technique are popular in recent years.

4) serially superadding the original test cases

Genetic algorithms are referred to as whole area searching because it begins from point group and not a single point. The original test case group is an important problem must be considered perfectly. We have found genetic algorithms to be powerful in locating test sets to cover disconnected subdomain comprising single test sets. These are difficult to find by other testing methods. But in the same time, the peculiarity greatly increases the use of time and space. In addition, it is difficult to proportionately apply the sets scale of initial test cases, the effect and the cost. If the set scale of initial test cases is great, the spending and the time will be great although the optimum in terms of high quality tests can be located. In the contrary, if the sets scale is small, the typicality and the representation of single test case will decrease.

Considering the multi-peak area can not be satisfied by function minimization method, we adopt the method of serially superadding the original test cases in order to increase the probability of whole area searching and avoid locating partial peak-value. The original test cases can be selected randomly. Only when the searching failed, the test cases will be superadded.

Assume that the ratio of successful searching is P_i if we superadd the i test case. The failing ratio is 1- P_i. So that when we superadd n times, the ratio of successful searching is:

$$P = 1 - \prod_{i=1}^{n} (1 - P_i)$$

Considering the test cases be superadded in the experiments, data selected as below:

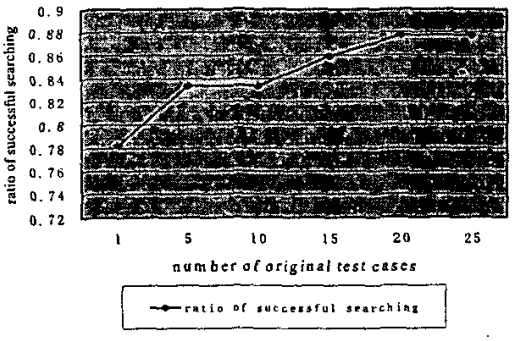

Fig.2 the efficiency of superadding the original test cases

Fig. 2 shows that if searching is fail, superaddition of original test cases can certainly improve the probability of successful searching. Because P_i may be 0, part of original test cases be superadded can be invalid.

In addition, in order to improve the searching efficiency, we generate original test cases by combining the method of manual and random. For different program units, we speculate the most possible centralized sub-area according to the type information of parameters and generate test cases in the area. For example, for integer parameters, the probability of the target value in the sub-area around the value 0 is more than the probability of it in the sub-area near the two end of the whole integer field. So that when we initialize integer parameters, we can generate test cases purposely in the sub-area around the value 0. At the same time, in applying the algorithm, it is important to ensure that the overlay of the population of original test sets is maintained, otherwise the global optimum in terms of high quality tests will not be located.

IV. CONCLUSION

Structural test data generation is all along a complicated problem. Methods and algorithms are not mature. The test data generation method of program execution shows its advantage because of the dynamic speciality. But as the key of this method, the searching algorithms can not be satisfied. The traditional function minimization method is good at finding the local optimum. But it is difficult to deal with the condition of multi-predicate and multi-peak space. The genetic algorithms is good at locating the test case in whole field, which can improve the searching algorithms. But there are some problems can not be resolved perfectly such as the mutation ratio and the original test case group.

This paper analysis the old algorithms, and discuss the methods to improve them. Consideration on some problem shows good effect in experiments. The goal we pursued is still how to advance the algorithms' efficiency and its practicality.

REFERENCES

[1] Korel, B. Automated software test data generation, TEEE Trans. Softw. Eng, 10 (6), 795-803, Nov, 1990.

[2] Akos Hajnal and Istvan Forgacs, An Applicable Test Data Generation Algorithm for Domain Errors, ACM Trans. Softw. Eng., 1998.

[3] B.F.Jones H.H.Sthamer and D.E.Eyres. Automatic structural testing using genetic algorithms. Software Engineering journal, 1996,9.

Author : Wan Lin born in 1974. She had her master degree in 1999, and now is reading for her doctor degree in Beijing armored force engineering institute. Search direction is software engineering and software testing.

E_mail: www_l@163.net

Phone number: 010-66719315 (office)