Published in IET Software
Received on 23rd November 2009
Revised on 5th February 2010
doi: 10.1049/iet-sen.2009.0092

ISSN 1751-8806

Automatic detection of infeasible paths in software testing

D. Gong[1] *X. Yao*[1,2]
[1]*School of Information and Electrical Engineering, China University of Mining and Technology, Xuzhou, Jiangsu 221116, People's Republic of China*
[2]*College of Science, China University of Mining and Technology, Xuzhou, Jiangsu 221116, People's Republic of China*
E-mail: yxjcumt@126.com

Abstract: A challenging problem in path-oriented test data generation is the presence of infeasible paths. Timely detecting these infeasible paths cannot only save test resources but also improve test efficiency. A popular method of detecting infeasible paths is to determine branch correlations, which is a difficult task and usually cannot be done timely and exactly. In this study, the authors propose a method of automatically determining the branch correlations of different conditional statements, therefore detecting infeasible paths. First, some theorems are given to determine branch correlations based on the probabilities of the conditional distribution corresponding to different branches' outcome (i.e. true or false); then, the maximum likelihood estimation is employed to obtain the values of these probabilities; finally, infeasible paths are detected according to branch correlations. The authors apply the proposed method in some typical programs, and the results show that the proposed method can accurately detect infeasible paths. The achievement provides an effective and automatic method of detecting infeasible paths, which is significant in improving the efficiency of software testing.

1 Introduction

One of the approaches to improving the quality of software is to do a large number of tests before delivery and usage in order to detect faults or errors. Software testing is an expensive, tedious and labour-intensive task, and accounts for up to 50% of the total cost of software development [1]. If the process can be performed automatically, it will undoubtedly shorten the period of software development and improve the quality of software, so as to enhance the market competitiveness.

Shan *et al.* [2] proposed that many software testing problems come down to the problem of path-oriented test data generation, which can be described as follows: for a given path of the program under test, search for a test datum in the input domain of the program, such that the traversed path of the test datum is just the target one.

A challenging problem with path-oriented test data generation is the presence of infeasible paths, that is there is no input datum for these paths to be executed.

Experimental evidences have shown that a significant amount of infeasible paths are present in a complicated program, and the detection of these infeasible paths is an undecidable question [3]. To the best of our knowledge, there has been no approach to successfully detecting a large number of infeasible paths of the program. The previous methods of detecting infeasible paths can be classified into two categories, that is static analysis and dynamic techniques, where the former is mainly based on the symbolic execution [4, 5] and the static branch correlations [6], providing partial solutions for some particular cases.

Generally speaking, dynamic techniques cannot distinguish the feasibility of paths directly. A common strategy in dynamic techniques is to limit the number and the depth of search. If the datum that traverses a target path has not been found in the final period of the search, the path will be considered infeasible [7].

The main cause of existing infeasible paths is the correlation of some conditional statements. There is at least one infeasible path in a program for each pair of correlated conditional statements, so studying the correlation of conditional statements plays an important role in detecting infeasible paths, whereas it is a difficult task and cannot be done timely and exactly. In fact, only about 13% of correlated conditional statements can be timely detected during compilation [8].

In this paper, we propose a novel method of automatically determining branch correlations, accordingly detecting infeasible paths in software testing. This method combines the advantages of static analysis and dynamic techniques, and determines the branch correlations of different conditional statements employing the maximum likelihood estimation. First, some theorems are given to determine branch correlations based on the probabilities of the conditional distribution corresponding to different branches' outcome (i.e. true or false); then, the values of the probabilities are obtained by the maximum likelihood estimation; finally, infeasible paths are detected according to branch correlations. We apply the proposed method to detect infeasible paths of some typical programs, and the results show its efficiency.

This paper is divided into seven sections, and the remainder is organised as follows: Section 2 reviews the related works on the detection of infeasible paths; Section 3 introduces some terminologies used in this paper; Section 4 expounds our method of detecting infeasible paths in detail; a case study and some experiments are given in Sections 5 and 6, respectively; finally, Section 7 draws conclusions and points out some possible research opportunities.

2 Related works

The cause and effect of infeasible paths of a program were discussed early by Hedley *et al.* [3]. In their work, the methods for writing a program, which do not contain any infeasible paths are also discussed. Bodik *et al.* [8] pointed out that the main cause of infeasible paths of a program is the correlation of some conditional statements. They also shown by experiences that 9–40% of conditional statements in a complicated program have correlations.

There are mainly two methods of detecting infeasible paths, that is static analysis and dynamic techniques.

2.1 Static analysis

Static analysis for detecting infeasible paths is mainly based on the symbolic execution and the static branch correlations.

The symbolic execution-based approach to detecting infeasible paths was proposed by Coward *et al.* [5] and [4, 9, 10]. A path is represented by a set of equations and the equations are solvable if and only if there are some inputs that drive the execution of the program down to the path. An infeasible path is generally detected when these equations are inconsistent with each other. However, the above approach is expensive; further more, it cannot behave well for a complicated program with loops, arrays, pointers as well as procedure calls.

The purpose of branch correlations is to determine the feasibility of a target path by investigating the branch correlation of different conditional statements. Bodik *et al.* [8] presented an algorithm for detecting infeasible paths incorporating the branch correlation analysis into the data flow analysis technique, which improves the precision of traditional data flow analysis. Chen *et al.* [11] proposed a method of detecting infeasible paths by investigating whether there exists a conflict between the assignment statement and the branch statement, as well as that among different branch statements.

Although the branch correlation of some conditional statements can be easily determined, it is a difficult task and cannot be done timely and exactly for many cases. It is reported that only about 13% of the analysable conditional statements show some correlations during compilation [8].

2.2 Dynamic techniques

As mentioned in the foregoing section that if the datum that traverses a target path has not been found in the final period of the search, the path will be considered infeasible.

A metric to predict the feasibility of a path and a strategy of generating its test data were proposed by Malevris *et al.* [12, 13]. They observed that the more the number of conditional statements contained in a path, the greater the probability of the path being infeasible. Ngo *et al.* [14] also proposed a heuristics-based approach to detecting infeasible paths, which is based on the observation that many infeasible paths exhibit some common properties. Through realising these properties in executing traces during the test data generation, infeasible paths can be detected.

A number of auxiliary tools and systems have been developed for infeasible path detection. Bueno *et al.* [15, 16] presented a tool that uses genetic algorithms to detect potential infeasible paths during the test data generation. They proposed a fitness function that combines both control flow and data flow information to better guide the search. However, the computational cost of this method is huge. In addition, Balakrishnan *et al.* [17] presented a syntactic language refinement technique that automatically excludes semantical infeasible paths from a program.

3 Terminology

In this section, we review some technical terms that will be used in the subsequent sections. The structure of a program P can be represented by a control flow graph, and denoted as follows

$$G(P) = \{N, E, s, e\}$$

where N is a set of nodes, E is a set of edges, s is the unique entry node and e is the unique exit node. A node $n \in N$ represents a statement of P. An edge $(n_i, n_j) \in E$ represents the control flow from statement n_i to statement n_j.

A path of P, denoted as Γ, is a sequence of nodes, that is $n_1, n_2, \ldots, n_k$, such that there is an edge from node n_i to n_{i+1}, where $i = 1, 2, \ldots, k-1$.

An input variable of P is a variable that appears in an input statement or an input parameter. If the input variables of P are $x_1, x_2, \ldots, x_n$, respectively, then the vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ will be called the input vector of P; accordingly, if the domain of x_i is D_i, where $i = 1, 2, \ldots, k-1$, then the input domain of P can be denoted as $D(P) = D_1 \times D_2 \times \cdots \times D_n$, where '$\times$' denotes an outer product operator.

A path Γ is called feasible, if there is an input datum $\boldsymbol{x}_0 \in D(P)$, such that the traversed path of $\boldsymbol{x}_0$ is just Γ; otherwise, if for $\forall \boldsymbol{x} \in D(P)$, the traversed path of $\boldsymbol{x}$ is not Γ, then Γ will be called infeasible. The presence of infeasible paths is mainly from different conditional statements contained in a path correlating to each other.

For the sake of simplicity, we say that the outcome of a conditional statement is true (or false) indicates its predicate being true (or false).

Let u and v be nodes in $G(P)$, we say that v post-dominates u if any directed path from u to e has to pass through v; we say that v is control dependent on u if and only if the following two conditions hold: (i) there exists a directed path Γ from u to v, such that except u and v, v post-dominates every node w in Γ; and (ii) v does not post-dominate u.

Assume that n_i and n_j are two conditional statements and n_j post-dominates n_i. If the outcome of n_i is true, and that of n_j must be true, then we say that (n_i, n_j) have true $\mapsto$ true (for short, $T \mapsto T$) correlation; on the contrary, if the outcome of n_i is true, and that of n_j must be false, then we say that (n_i, n_j) have true $\mapsto$ false (for short, $T \mapsto F$) correlation. Similarly, the concept of false $\mapsto$ true and false $\mapsto$ false (for short, $F \mapsto T$ and $F \mapsto F$, respectively) correlations of (n_i, n_j) can also be defined.

The above concepts can be generalised to the branch correlations of more than two conditional statements. Let $(n_1, n_2, \ldots, n_m)$ be m conditional statements, where n_{i+1} post-dominates or is control dependent on n_i for all $1 \le i \le m-1$. If the outcome of $n_1, n_2, \ldots, n_{m-1}$ is $\varphi_1, \varphi_2, \ldots, \varphi_{m-1}$, where $\varphi_i \in$ {true, false}, and that of n_m must be φ_m, where $\varphi_m \in$ {true, false} once n_m is executed, then we say that $(n_1, n_2, \ldots, n_m)$ have $\varphi_1, \varphi_2, \ldots, \varphi_{m-1} \mapsto \varphi_m$ correlation.

4 Method of automatically detecting infeasible paths

In this section, we propose a method of automatically detecting infeasible paths. First, some sample values are obtained by executing the program under test, and the values of the statistics for estimating the conditional probability are calculated; then the type of branch correlations is determined according to the conditional probability; finally, infeasible paths are detected based on the type of branch correlations.

4.1 Determination of branch correlations

For convenience, we first consider the case of determining the correlation of two conditional statements, and then that of more than two conditional statements. Consider two conditional statements n_i and n_j of P, for the input vector $\boldsymbol{x}$ of P, suppose the above conditional statements are both executed. Therefore it is meaningful to define the following two variables:

$$X = \begin{cases} 1, & \text{if the outcome of } n_i \text{ on } \boldsymbol{x} \text{ is true} \\ 0, & \text{otherwise} \end{cases}$$

$$Y = \begin{cases} 1, & \text{if the outcome of } n_j \text{ on } \boldsymbol{x} \text{ is true} \\ 0, & \text{otherwise} \end{cases}$$

Because $\boldsymbol{x}$ is sampled stochastically on $D(P)$, both X and Y are stochastic variables, and obey (0,1)-distribution. In the following, we investigate the branch correlation of (n_i, n_j) employing the conditional distributions of X and Y.

Assume that

$$p_1 = p\{Y = 1|X = 1\}$$
$$p_2 = p\{Y = 1|X = 0\}$$

where $p\{Y = 1|X = 1\}$ denotes the conditional probability that Y is equal to one on condition of X being equal to one; similarly, $p\{Y = 1|X = 0\}$ denotes the conditional probability that Y is equal to one on condition of X being equal to zero. It is easy to observe that both p_1 and p_2 are constants between zero and one, and

$$p\{Y = 0|X = 1\} = 1 - p_1$$
$$p\{Y = 0|X = 0\} = 1 - p_2$$

We can obtain the following theorems

Theorem 1: If $p_1 = 1$, then (n_i, n_j) have $T \mapsto T$ correlation; If $p_1 = 0$, then (n_i, n_j) have $T \mapsto T$ correlation; If $0 < p_1 < 1$, then (n_i, n_j) have neither $T \mapsto T$ correlation nor $T \mapsto F$ correlation.

Theorem 2: If $p_2 = 1$, then (n_i, n_j) have $F \mapsto T$ correlation; If $p_2 = 0$, then (n_i, n_j) have $F \mapsto F$ correlation; If $0 < p_2 < 1$, then (n_i, n_j) have neither $F \mapsto T$ correlation nor $F \mapsto F$ correlation.

The proofs of these theorems are omitted because they can be easily obtained from the definition of these four kinds of correlations.

4.2 Estimation of p_1 and p_2

It can be observed from Theorems 1 and 2 that, in order to determine the correlation between n_i and n_j, we need to know the values of p_1 and p_2. In the following, we use maximum likelihood estimation to estimating their values.

Stochastically sampling M values $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_M$ of $\boldsymbol{x}$, assume that both n_i and n_j are executed on $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_M$ (if either n_i or n_j is not executed on $\boldsymbol{x}_i$, then we call $\boldsymbol{x}_i$ invalid and perform another sampling, until both of them are executed), we can obtain a set of samples with capacity being M, that is $(X_1, Y_1), (X_2, Y_2), \ldots, (X_M, Y_M)$. Denote their values as follows

$$(x_1, y_1), (x_2, y_2), \ldots, (x_M, y_M)$$

Construct the following statistics

$$\Lambda = \sum_{i=1}^{M} X_i, \quad \Upsilon_1 = \sum_{i=1}^{M} X_i Y_i, \quad \Upsilon_2 = \sum_{i=1}^{M} (1 - X_i) Y_i$$

with their values being λ, γ_1 and γ_2, respectively, where λ is equal to M means all M outcome of n_i are true; whereas λ is equal to zero means all M outcome of n_i are false. Generally speaking, λ is larger than zero and smaller than M. The estimates of p_1 and p_2 are given as follows.

Theorem 3: The maximum likelihood estimates of p_1 and p_2 are

$$\hat{p}_1 = \frac{\gamma_1}{\lambda}, \quad \hat{p}_2 = \frac{\gamma_2}{M - \lambda}$$

respectively.

Proof:

$$p\{Y = y | X = x\} = p_1^{xy} (1 - p_1)^{x(1-y)} p_2^{(1-x)y} (1 - p_2)^{(1-x)(1-y)}$$

where $x, y = 0, 1$. Hence the maximum likelihood function is

$$L(p_1, p_2) = \prod_{i=1}^{M} (p_1^{x_i y_i} (1 - p_1)^{x_i(1-y_i)} \cdot p_2^{(1-x_i)y_i} (1 - p_2)^{(1-x_i)(1-y_i)})$$

thus

$$\ln(L(p_1, p_2)) = \ln(p_1) \sum_{i=1}^{M} x_i y_i + \ln(1 - p_1) \sum_{i=1}^{M} x_i (1 - y_i)$$
$$+ \ln(p_2) \sum_{i=1}^{M} (1 - x_i) y_i + \ln(1 - p_2) \sum_{i=1}^{M} (1 - x_i)(1 - y_i)$$

let

$$\begin{cases} \dfrac{\partial \ln(L(p_1, p_2))}{\partial p_1} = 0 \\ \dfrac{\partial \ln(L(p_1, p_2))}{\partial p_2} = 0 \end{cases}$$

we have

$$\hat{p}_1 = \frac{\sum_{i=1}^{M} x_i y_i}{\sum_{i=1}^{M} x_i} = \frac{\gamma_1}{\lambda}, \quad \hat{p}_2 = \frac{\sum_{i=1}^{M} (1 - x_i) y_i}{M - \sum_{i=1}^{M} x_i} = \frac{\gamma_2}{M - \lambda}$$

We execute P based on some inputs, and obtain the estimates of p_1 and p_2, then determine whether n_i and n_j have correlations, if have, what kinds of correlations according to Theorems 1 and 2.

The above method can be generalised to the case of determining the branch correlation of more than two conditional statements. Let $(n_1, n_2, \ldots, n_m)$ be m conditional statements, and choose $\boldsymbol{x} \in D(P)$ so that all these conditional statements can be executed on $\boldsymbol{x}$. Define

$$X_k = \begin{cases} 1, & \text{if the outcome of } n_k \text{ on } \boldsymbol{x} \text{ is true} \\ 0, & \text{otherwise} \end{cases} \qquad k = 1, 2, \ldots, m$$

then we obtain a stochastic vector $(X_1, X_2, \ldots, X_m)$. Repeating this process M times using the same method, we obtain a set of samples with capacity being M, that is $(X_{i1}, X_{i2}, \ldots, X_{im})$, $i = 1, 2, \ldots, M$.

calculate the following statistic

$$\hat{p} = \frac{\sum_{i=1}^{M} x_{i1}^{\delta_1} (1 - x_{i1})^{1-\delta_1} \cdots x_{im}^{\delta_m} (1 - x_{im})^{1-\delta_m}}{\sum_{i=1}^{M} x_{i1}^{\delta_1} (1 - x_{i1})^{1-\delta_1} \cdots x_{i,m-1}^{\delta_{m-1}} (1 - x_{i,m-1})^{1-\delta_{m-1}}}$$

where

$$\delta_i = \begin{cases} 1 & \text{if } \varphi_i = \text{true} \\ 0 & \text{otherwise} \end{cases}$$

If $\hat{p} = 1$, then $(n_1, n_2, \ldots, n_m)$ have $(\varphi_1, \varphi_2, \ldots, \varphi_{m-1} - \varphi_m)$ correlation. □

4.3 Detection of infeasible paths

Once we have knowledge of the correlation and the type of different conditional statements, we can detect infeasible paths of a program according to the following theorem.

Theorem 4: Suppose n_i and n_j are two condition statements of P. If (n_i, n_j) have $T \mapsto T$ (or $T \mapsto F$) correlation, then any path including the true branch of n_i and the false (or true) branch of n_j will be infeasible; similarly, if (n_i, n_j) have $F \mapsto T$ (or $F \mapsto F$) correlation, then any path including the false branch of n_i and the false (or true) branch of n_j will be infeasible.

Proof: If (n_i, n_j) have $T \mapsto T$ correlation, then any feasible path including the true branch of n_i must also include the true branch of n_j, which means that any path including the true branch of n_i and the false branch of n_j will be infeasible. The other cases can also be proved using the same methods. □

4.4 Notes on presence of loops

Most of real-world programs include loops. If the number of cycles is small (not more than three), then we will regard different times of cycles as different parts of the program, for example, for the following loop:

```
1   for (i = 1; i <= 2; i ++)

      {

2       if( ··· )

3           ······

      }
```

In this loop, Statement 2 is a conditional statement. The control flow graph corresponding to this loop is shown as follows: where 2(1) and 2(2) mean the first and the second time of executing statement 2, respectively.

If the number of cycles is large, then we will artificially control it to not more than three.

5 Case study

In this section, we explain and validate the method proposed in this paper through an example C program, denoted as P. Fig. 1 shows its source code and control flow graph. The statements of P are labelled as ①∼⑨.

P includes three conditional statements that are labelled as ①, ④ and ⑦, respectively. The aim of this experiment is to investigate the branch correlations of the three condition statements employing our method. The input vector of P is (A, B) with its domain being $D(P) = [-500, 500]^2$.

Step 1: Estimation of conditional probabilities: First, define three stochastic variables X_1, X_2 and X_3 corresponding to the outcome of conditional statements ①, ④ and ⑦, respectively, and let

$$p_i = p\{X_2 = 1 | X_1 = i\}, \; i = 0, 1$$

$$p'_i = p\{X_3 = 1 | X_2 = i\}, \; i = 0, 1$$

$$p''_i = p\{X_3 = 1 | X_1 = i\}, \; i = 0, 1$$

Then, estimate the values of the above three parameters based on sampling. Stochastically sampling M input vectors in $D(P)$. In general, in order to obtain representative samples, M should be large enough, and the M input vectors should distribute in various regions of $D(P)$. Here, the value of M is set to 50. Therefore a set of samples $\{(X_{i1}, X_{i2}, X_{i3}), i = 1, 2, \ldots, 50\}$ can be obtained. In order to obtain the values of these samples, the original program should be instrumented appropriately. Fig. 2 shows the instrumented program of P.

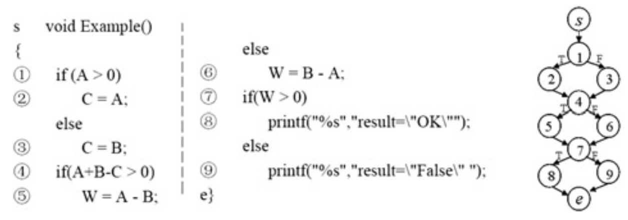
Finally, estimate the values of p_i, p'_i and p''_i by Theorem 3. Table 1 lists these estimates.

In order to investigate the branch correlations of three conditional statements, define

$$p_{ij} = p\{X_3 = 1 | X_1 = i, X_2 = j\}, \quad i, j = 0, 1$$

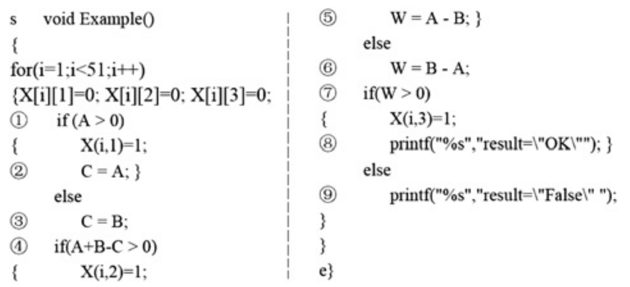
Employing the same approach as the above, the estimates of these conditional probabilities can also be obtained. Table 2 lists these estimates.

Step 2: Determination of branch correlations: We can observe from Theorems 1 and 2 and Table 1 that conditional statements (①, ④) have $F \mapsto F$ correlation because of $\hat{p}_0$ being equal to zero; nevertheless, conditional statements (④, ⑦) and (①, ⑦) have no branch correlation because the values of p'_i and $p''_i (i = 0, 1)$ are all larger than zero as well as smaller than one.



```
s   void Example()                  |
{                                   |       else
①     if (A > 0)                    |  ⑥       W = B - A;
②         C = A;                    |  ⑦   if(W > 0)
      else                          |  ⑧       printf("%s","result=\"OK\"");
③         C = B;                    |       else
④     if(A+B-C > 0)                 |  ⑨       printf("%s","result=\"False\" ");
⑤         W = A - B;                |  e}
```

Figure 1 *Source code and control flow graph of P*



```
s   void Example()                          |  ⑤       W = A - B; }
{                                           |       else
for(i=1;i<51;i++)                           |  ⑥       W = B - A;
{X[i][1]=0; X[i][2]=0; X[i][3]=0;           |  ⑦   if(W > 0)
①     if (A > 0)                            |  {       X(i,3)=1;
{         X(i,1)=1;                         |  ⑧       printf("%s","result=\"OK\""); }
②         C = A; }                          |       else
      else                                  |  ⑨       printf("%s","result=\"False\" ");
③         C = B;                            |  }
④     if(A+B-C > 0)                         |  }
{         X(i,2)=1;                         |  e}
```

Figure 2 *Instrumented program of P*

Table 1 Estimates of p_i, p_i' and p_i''

parameter	$\hat{p}_1$	$\hat{p}_0$	$\hat{p}_1'$	$\hat{p}_0'$	$\hat{p}_1''$	$\hat{p}_0''$
estimate	0.4815	0	0.6154	0.5135	0.2963	0.8261

Table 2 Estimates of p_{ij}

parameter	$\hat{p}_{00}$	$\hat{p}_{01}$	$\hat{p}_{1,0}$	$\hat{p}_{1,1}$
estimate	0.826	∞	0	0.615

For the case of three conditional statements, we can observe from the generalisation of Theorems 1 and 2 as well as Table 2 that conditional statements (①, ④, ⑦) are $TF \mapsto F$ correlative because of $\hat{p}_{10}$ being equal to zero; furthermore, because (①, ④) are $F \mapsto F$ correlative, $\hat{p}_{01}$ has no practical meaning, and the result of Table 2 also verify this fact.

Step 3: Detection of infeasible paths: Based on these branch correlations of different conditional statements, we detect the following infeasible paths of P

Γ_1=(①, ③, ④, ⑤, ⑦, ⑧)

Γ_2=(①, ③, ④, ⑤, ⑦, ⑨)

Γ_3=(①, ②, ④, ⑥, ⑦, ⑧)

6 Experiments

In this section, we validate the performance of our method through a group of experiments.

6.1 *Experiments on two benchmark programs*

In this subsection, we investigate the branch correlations of different conditional statements and detect the infeasible paths of two benchmark programs, that is the triangle classification program, denoted as P_1 and the bubble sort program, denoted as P_2.

6.1.1 Detection of infeasible paths of P_1: Fig. 3

shows the source code and control flow graph of P_1 compiled in C Language. The input variables are a, b and c with their domains being integers between zero and 300. We obtain the values of input variables by means of probability selection to guarantee the sampling to be equilateral or isosceles with some probabilities. In this experiment, let the probability of the event that can construct an equilateral or an isosceles triangle be 0.1, and the capacity of the set of samples be 100. This program includes six conditional statements. Employing the method proposed in Section 4, we obtain the branch correlation of these conditional statements, listed as Table 3.

What to explain is that because some correlations can be derived from other ones, these correlations are not listed in Table 3, for example, (13,16,18) have $FT \mapsto F$ correlation that can be derived from the $T \mapsto F$ correlation of (16,18). We need not list it again because all infeasible paths can be detected by the former can also be detected by the latter.

Now we detect the infeasible paths of P_1 based on the results of Table 3.

Because (16, 18) are $T \mapsto F$ correlative, we know from Theorem 4 that any path including the true branch of statement 16 and the true branch of statement 18 is infeasible. Thus the following infeasible paths can be

Table 3 Correlations of different conditional statements of P_1

(16,18)	$T \mapsto F$	(5,13,16)	$TF \mapsto F$
(1,5,9)	$TT \mapsto T$	(9,13,16)	$TF \mapsto F$
(1,13,16)	$TF \mapsto F$	(1,5,13,18)	$TTF \mapsto F$
(5,9,13)	$TF \mapsto T$	(5,9,13,18)	$TTF \mapsto F$, $TFF \mapsto T$

```
s     void Triangle(int a, int b, int c)
1     { if (a > b)
2       {   t=a;
3           a=b;
4           b=t; }
5       if (a >c)
6       {   t=a;
7           a=c;
8           c=t; }
9       if (b > c)
10      {   t=b;
11          b=c;
12          c=t; }
13    if (a+b<=c)
14        strcpy(Type, "NOT_A_TRIANGLE");
      else
15    {   strcpy(Type, "SCALENE");
16        if(a==b&&b==c)
17            strcpy(Type, "EQUILATERAL");
18        if((a==b||b==c)&& a!=c)
19            strcpy(Type, "ISOSCELES");    }
e     }
```

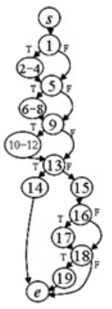


Figure 3 *Source code and control flow graph of P_1*

detected based on this correlation

$$\Gamma_1 = (s, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, e)$$

$$\Gamma_2 = (s, 1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 15, 16, 17, 18, 19, e)$$

$$\Gamma_3 = (s, 1, 2, 3, 4, 5, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, e)$$

$$\Gamma_4 = (s, 1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, e)$$

$$\Gamma_5 = (s, 1, 2, 3, 4, 5, 9, 13, 15, 16, 17, 18, 19, e)$$

$$\Gamma_6 = (s, 1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, e)$$

$$\Gamma_7 = (s, 1, 5, 9, 13, 15, 16, 17, 18, 19, e)$$

Employing the same method as the above, we can detect other infeasible paths of P_1. Table 4 lists the statistic result of infeasible paths.

In Table 4, the first column means the number of paths of P_1, which is the sum of that of feasible and infeasible ones; $I_{\mathtt{total}}$ refers to the number of infeasible paths; IBDC$_i$ means the number of infeasible paths detected by the branch correlations of i conditional statements. $D_{\mathtt{total}}$ means the number of infeasible paths detected by our method, and is equal to the sum of IBDC$_2$, IBDC$_3$ and IBDC$_4$. We can observe from Table 4 that all infeasible paths of P_1 can be detected employing our method.

6.1.2 *Detection of infeasible paths of P_2*:

Fig. 4 shows the source code and control flow graph of P_2 compiled in C Language. The input vector is Array [4], an array with four integers whose domain is [0, 100]. This program is complicated because it includes a while loop, a nested for loop and a nested if statement. Since the number of executing Statement 6 is fixed, we do not consider it as a conditional statement here. But we consider Statement 4 as a conditional statement because the number of executing it depends on its conditions. In order to easily understand the structure of P_2, Fig. 4 shows its control flow graph, where Num(i) means the i-th execution of statement Num.

Table 4 Statistic result of infeasible paths of P_1

No. of paths	$I_{\mathtt{total}}$	Infeasible paths detected by our method			
		IBDC$_2$	IBDC$_3$	IBDC$_4$	D_{total}
40	**22**	8	11	3	**22**

Employing the same method as the foregoing subsection, we can obtain the correlations of different conditional statements of P_2, listed as Table 5, and further detect its infeasible paths, whose statistic result is listed as Table 6.

The only one infeasible path that we have not detected is $\Gamma = (s, 1, 2, 3, 4(1), e)$. Because the conditions always satisfy when Statement 4 is first executed, the outcome of 4(1) cannot be false. This infeasible path cannot be detected by our method.

6.2 *Experiments on real-world programs*

In order to validate the performance of our method in detecting infeasible paths of a complicated program, we further conduct another group of experiments on real-world programs.

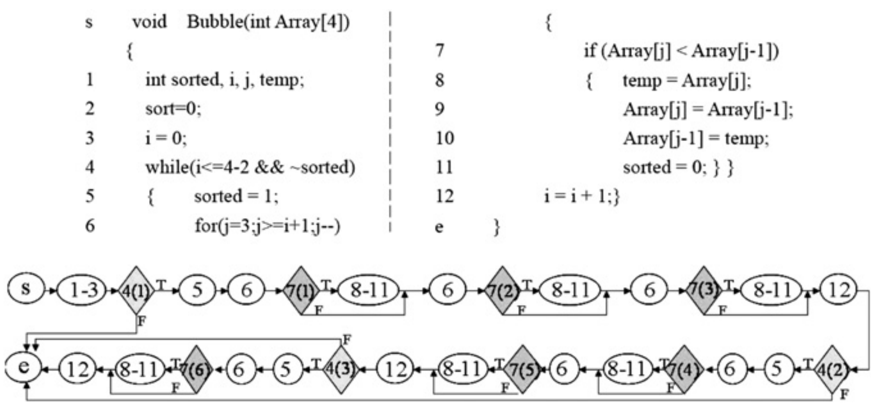
We select some functions stochastically from four programs that are downloaded from the system 'Software-artifact Infrastructure Repository' for experiments [18]. Table 7 lists their descriptions in detail.

There are a large number of conditional statements in these programs and many of which have correlations, for example, a fragment of the program 'space' is as follows.

```
. . .. .. .. . .

error = adddef( * curr_ptr, curr_ptr, addrem_ptr)

if (error == 0) {
```



```
s    void   Bubble(int Array[4])
     {
1      int sorted, i, j, temp;
2      sort=0;
3      i = 0;
4      while(i<=4-2 && ~sorted)
5      {      sorted = 1;
6             for(j=3;j>=i+1;j--)
              {
7                 if (Array[j] < Array[j-1])
8                 {     temp = Array[j];
9                       Array[j] = Array[j-1];
10                      Array[j-1] = temp;
11                      sorted = 0; } }
12            i = i + 1;}
e    }
```

Figure 4 *Source code and control flow graph of P_2*

Table 5 Correlations of different conditional statements of P_2

(7(1), 4(2))	$T \mapsto T$
(7(2), 4(2))	$T \mapsto T$
(7(3), 4(2))	$T \mapsto T$
(7(4), 4(3))	$T \mapsto T$
(7(5), 4(3))	$T \mapsto T$
(7(1), 7(2), 7(4))	$TF \mapsto T$
(7(2), 7(3), 7(4))	$FT \mapsto F$
(7(2), 7(3), 7(5))	$TF \mapsto F$
(7(2), 4(2), 7(4))	$FT \mapsto F$
(7(3), 7(4), 7(5))	$FF \mapsto F$, $FT - F$
(7(3), 7(4),4(3))	$FF \mapsto F$
(7(3), 7(4), 7(6))	$FT \mapsto F$
(7(3), 4(3), 7(6))	$FT \mapsto F$
(7(4), 7(5), 4(3))	$FF \mapsto F$, $TF \mapsto T$
(7(4), 7(5), 7(6))	$TF \mapsto F$
(7(5), 4(3),7(6))	$FT \mapsto F$
(7(1), 7(2), 7(3), 4(2))	$FFF \mapsto F$

Table 6 Statistic result of infeasible paths of P_2

No. of paths	I_{total}	Infeasible paths detected by our method			
		IBDC$_2$	IBDC$_3$	IBDC$_4$	D_{total}
105	81	31	48	1	80

Table 7 Descriptions of programs under test

Program	No. of selected functions	Line of codes
space	3	272
replace	3	162
flex	5	427
sed	5	585

```
    * pp2 =  * curr_ptr;

    return 0;

  };

  if (error == 17) {

    printf('\ n % s', ErrorMessages[62])

    return 17

  }

    if (error == 1) {

      error = remdef( * curr_ptr, curr_ptr, addrem_ptr)

  }

  .........
```

In this fragment, any pair of these three conditional statements have $T \mapsto F$ correlation. So any path including more than two true branches of these three conditional statements is infeasible.

For each program under test, we determine the correlations of two, three and four conditional statements, and then detect its infeasible pathes according to these correlations. Table 8 lists the statistic result of infeasible paths of these programs, where BC$_i$ means the number of correlated conditional statements that includes i conditional statements ($i = 2$, 3, 4), and

$$R_s = \frac{D_{\text{total}}}{I_{\text{total}}} \times 100\%$$

which is the ratio of the number of infeasible paths detected by our method to that of infeasible ones of the program.

We can observe from Table 8 that, for 'replace', 'flex' and 'sed', we have detected all infeasible paths; but for 'space', although we have not detected all infeasible paths, R_s is as large as 99.81%. The reason that we cannot detect all

Table 8 Statistic result of infeasible paths of real-world programs

program		space	replace	flex	sed
no. of paths		735	976	397	821
I_{total}		**525**	**719**	**318**	**631**
Branch correlations	BC$_2$	12	26	14	36
	BC$_3$	37	23	12	31
	BC$_4$	6	7	23	12
infeasible paths detected by our method	IBDC$_2$	162	238	98	296
	IBDC$_3$	314	409	189	250
	IBDC$_4$	48	72	31	85
	D_{total}	524	719	318	631
R_s(%)		99.81	100	100	100

infeasible paths lies in the program itself: there are correlations of more than four conditional statements.

To sum up, our method can detect almost all infeasible paths of benchmark programs as well as some real-world programs, therefore, it is an effective method of detecting infeasible paths of a program.

7 Conclusions

Software testing is very important to improve the quality of software. Timely detecting infeasible paths can not only save test resources but also improve test efficiency. Up to present, there has been few achievement available to detect infeasible paths timely and exactly.

We propose a method of detecting infeasible paths based on the correlations of different conditional statements. The correlation and the kind of correlations are determined according to the conditional probabilities which are obtained employing the maximum likelihood estimation.

We show the process of detecting infeasible paths through a case study in detail, which includes three steps. Further, we apply the proposed method in some typical programs, and the experimental results show that our method can accurately detect almost all infeasible paths.

To the best of our knowledge, it is the first time that detecting infeasible paths based on the correlations of different conditional statements. Our achievement provides an effective and automatic method of detecting infeasible paths, which is significant in improving the efficiency of software testing.

It should be noted that our experimental results are still preliminary. In our future study, we will apply our method to more complicated programs to validate its effectiveness; in addition, how to efficiently sampling the input domain so that the samples can sequentially execute the desired conditional statements is also our future research topic.

8 Acknowledgments

This study was jointly funded by the Six Peak Talents Projects of Jiangsu Province with grant no. 2008125 and the Youth Research Funds of China University of Mining and Technology with grant no. 2008A034.

9 Reference

[1] BEIZER B.: 'Software testing techniques' (International Thomson Computer Press, 1990)

[2] SHAN J., JIANG Y., SUN P.: 'The progress of testing research', *J. Beijing Univ. (Nat. Sci.)*, 2005, **41**, (1), pp. 134–145

[3] HEDLEY D., HENNELL M.A.: 'The causes and effects of infeasible paths in computer programs'. Proc. Eighth Int. Conf. on Software Engineering, 1985, pp. 28–30

[4] ZHANG J., WANG X.: 'A constraint solver and its application to path feasibility analysis', *Int. J. Softw. Eng. Knowledge Eng.*, 2001, **11**, pp. 139–156

[5] COWARD P.D.: 'Symbolic execution and testing', *Inf. Softw. Technol.*, 1991, **33**, (1), pp. 53–64

[6] FORGACS I., BERTOLINO A.: 'Feasible test path selection by principal slicing'. Proc. Sixth Eur. Software Engineering Conf. and Fifth ACM SIGSOFT Symp. on the Foundations of Software Engineering, 1997, pp. 378–394

[7] SITTISAK S., CHIDCHANOK L., PERAPHON S.: 'An address mapping approach for test data generation of dynamic linked structures', *Inf. Softw. Technol.*, 2005, **47**, pp. 199–214

[8] BODIK R., GUPTA R., SOFFA M.L.: 'Refining data flow information using infeasible paths'. Proc. Sixth Eur. Softw. Eng. Conf. and Fifth ACM SIGSOFT Symp. on the Foundations of Software Engineering, 1997, pp. 361–377

[9] ANTONELLA S., GIGLIOLA V.: 'Formula-based abstractions and symbolic execution for model checking programs', *Microprocessors Microsyst.*, 2004, **28**, pp. 69–76

[10] BAHMAN P., MARJAN S., HOSSEIN H., FARHAD A.: 'Automated analysis of reo circuits using symbolic execution', *Electron. Notes in Theor. Comput. Sci.*, 2009, **255**, pp. 137–158

[11] CHEN T., MITRA T., ROYCHOUDHURY A., SUHENDRA V.: 'Exploiting branch constraints without exhaustive path enumeration'. Proc. Fifth Int. Workshop on Worst-Case Execution Time Analysis, 2005, pp. 40–43

[12] MALEVRIS N.: 'A path generation method for testing LCSAJs that restrains infeasible paths', *Inf. Softw. Technol.*, 1995, **37**, pp. 435–441

[13] MALEVRIS N., YATES D.F., VEEVERS A.: 'Predictive metric for likely feasibility of program paths', *Inf. Softw. Technol.*, 1990, **32**, pp. 115–118

[14] NGO M.N., TAN H.B.K.: 'Heuristics-based infeasible path detection for dynamic test data generation', *Inf. Softw. Technol.*, 2008, **50**, pp. 641–655

[15] BUENO P.M.S., JINO M.: 'Automatic test data generation for program paths using genetic algorithms', *Int. J. Softw. Eng. Knowledge Eng.*, 2002, **12**, pp. 691–709

[16] BUENO P.M.S., JINO M.: 'Identification of potentially infeasible program paths by monitoring the search for test data'. Proc. 15th IEEE Int. Automated Software Engineering Conf., 2000, pp. 209–218

[17] BALAKRISHNAN G., SANKARANARAYANAN S., IVANČIĆ F., WEI O., GUPTA A.: 'SLR: path-sensitive analysis through infeasible-path detection and syntactic language refinement', (*LNCS*, **5079**), 2008, pp. 238–254

[18] SIR: 'A repository of software-related artifacts meant to support rigorous controlled experimentation', Available at: http://sir.unl.edu/portal/index.html