Computer and Information Science
www.ccsenet.org/cis
Vol. 3, No. 1
February 2010

A New Method of Reducing Pair-wise Combinatorial Test Suite

Lixin Wang (Corresponding author)
College of Information Science and Technology, Donghua University, Shanghai, China
2999 Renmin Road North, Songjiang District, Shanghai, 201620, China
&
School of Electronics and Information Engineering
Anhui Institute of Architecture and Industry, Hefei, China
856, Jinzhai Road, Baohe District, Hefei, Anhui, 230022, China
E-mail: wlx@mail.dhu.edu.cn

Renxia Wan
College of Information Science and Technology, Donghua University, Shanghai, 201620, China
E-mail: xxrrww2002@163.com

Abstract

The biggest problem for combinatorial test is a numerous number of combinations of input parameters by combinatorial explosion. Pair-wise combinatorial coverage testing is an effective method which can reduce the test cases in a suite and is able to detect about 70% program errors. But, under many circumstances, the parameters in programs under test (PUTs) have relations with each other. So there are some ineffective test cases in pair-wise combinatorial test suites. In this paper, we propose a method of reducing ineffective combinatorial test cases from pair-wise test suite. The main ideas of the method is that we firstly analyzes the dependent relationships among input parameters, then use the relationships to reduce ineffective pair-wise combinations of input parameters, and lastly generate the pair-wise combinatorial coverage test suite. The experiments show that the method is feasible and effective, and considerably reduce the number of pair-wise combinatorial test cases for some programs under test.

Keywords: Pair-wise combinatorial test, Test suite, Dependent relationships

1. Introduction

The combinatorial test is one of main approaches in black-box tests. The combinatorial test creates test suites by selecting values of input parameters and combining these parameter values as test cases(Mandl, 1985). These parameters include program configuration parameters, internal events, user inputs, and other external events (Shiba, et al. 2004). Since there are often too many parameter combinations, so it is impossible to test all possible combinations of input parameters for many programs under test (PUTs). Pair-wise combinatorial testing is an effective method which can decrease the number of test cases in a suite and is able to detect about 70% program errors. Many researchers have presented a variety of methods (Cohen, et al. 1997) (Shiba, et al. 2004) (Tai, et al. 2002) (Schroeder, et al. 2000) to generate pair-wise combinatorial test suites and developed over ten tools to generate pair-wise test cases (Czerwonka.2009). But these methods do not consider the relationships of input parameters for a PUT.

In the definition of combinatorial test, a precondition is that the input parameters of a PUT are independent with each other. But, under many circumstances, these parameters have relations with each other and we often need test the PUT using combinatorial testing. So there are many invalid pair-wise combinations of input parameters due to the input parameters relationships so as to have some redundant and ineffective test cases in test suite, and degrade the performance of algorithm for generating test suite. So it is necessity to find some methodology to reduce the pair-wise combinatorial test suite.

In this paper, we present a new method to remove the ineffective pair-wise test cases from combinatorial test suite. The method is based on the using the dependent relationships of input parameters to achieve the aim of reducing pair-wise combinatorial test suite. This paper is constructed as follows: Section 2 analyzes the ineffective test cases in test suite.

Section 3 describes how to obtain the pair-wise combinations of input parameters. Section 4 elaborates the process of reducing the pair-wise combinatorial test suite. The experiment results and analysis are given in section 5, and the conclusions are presented in section 6.

2. Analysis of the ineffective pair-wise combinatorial test cases

2.1 The pair-wise combinatorial test

In the combinatorial test method, we generates test suite that cover all pair-wise, triple, or *n*-way combinations of test parameters specified in formal test requirements.

We usually generate pair-wise coverage test suite to test PUTs. Kuhn D. R. et al. found that about 70% field faults were caused by either incorrect single values or by an interaction of pairs of values in an empirical study of user interface software. David M. Cohen in their code coverage study also indicated that pair-wise coverage is sufficient for good code coverage (Cohen, et al. 1997).

See Table I, Cohen et al. tried several different models for the sort command. Table I shows the coverage data for two of them. The models differed from each other in two ways. First, the number of parameters were varied, which is called width, and the maximum number of values for a parameter, which is called height. In general, the number of pair-wise test cases is at least the product of the number of values for the two parameters with the most values. Consequently, rewriting a model to decrease the maximum number of values per parameter can decrease the number of tests, even if it increases the number of parameters.

Pair-wise combination is that for any two parameters p_1 and p_2 and any valid values v_1 for p_1 and v_2 for p_2, combine the v_1 and v_2 as a element into a set of combination. Covering all pair-wise combinations means that for any two parameters p_1 and p_2 and any valid values $v1$ for p_1 and v_2 for p_2, there is a test suite in which p_1 has the value v_1 and p_2 has the value v_2. We call the test suite as pair-wise combinatorial test suite. We use the pair-wise combinatorial test cases as input to test the PUT, which is called pair-wise combinatorial coverage test.

But, since there are dependent relationships between input parameters, the set of pair-wise combination often have some ineffective test cases. How do we find these ineffective test cases? And how do we reduce these ineffective test cases?

2.2 Ineffective test cases in pair-wise combinatorial test suite

Sometimes we need use combinatorial testing to test some PUTs with dependent relationships of input parameters. So there will be some ineffective pair-wise test cases in test suite.

For example, with respect to a PUT, we test an inquiry interface of the PUT, which has year, month, day, and other selections. After we select February (not other values of months), there are 1, 2, …, 28 or 29 in the day item list, but there will be not 30 and 31. In other words, we do not obtain some test cases such as (year-number, February, 30) and (year-number, February, 31), since there have been some relative restrictions among this input parameters in program codes, and some relative tests may have been conducted during white-box test or component test. So, in system or integrate test, we can find that the values of input parameter days are depend on the values of input parameter month in the PUT, and the combinations of (year-number, February, 30) and (year-number, February, 31) are ineffective.

Above example shows that a dependent relationship between two input parameters exists when the value of one input parameter implies some restrictions on the values of another input parameter. In this case, some input parameter combinations are not allowed and the total space of input parameter combination can be partitioned into the allowed and not allowed.

Definition 1: Suppose x_i and x_j are two input parameters, $i<j$, $a\in x_i$, $b\in x_j$. The pair-wise combination (a,b) is ineffective if and only if the (a,b) can not satisfy the user's demands, don't exist or is not able to be obtained.

For a pair-wise combination set with some ineffective combinations, when we generate the test suite which covers the combination set, the test suite will include some ineffective test cases. The definition of the ineffective test case is as Definition 2.

Definition 2: Suppose x_i and x_j are two input parameters, $i<j$, $a\in x_i$, $b\in x_j$ and (a,b) is a ineffective pair-wise combination, if a test case t denotes as $(v_1,v_2,\ldots, v_i,\ldots, v_j,\ldots,v_n)$, and $v_i=a$ and $v_j=b$, then we call the t is an ineffective pair-wise combinatorial test case.

In above-mentioned example, if a test case includes (year-number, February, 30) or (year-number, February, 31), then the test case is an ineffective combinatorial test case; if a test case includes (February, 30) or (February, 31), then the test case is an ineffective pair-wise combinatorial test case. For the pair-wise combinatorial coverage test, these ineffective pair-wise combinatorial test cases must reduce from test suite. But if we reduce these ineffective test cases in the manual analysis way, this will be very troublesome. In some complex cases, it may not be able to find all of these ineffective test cases. Therefore, we must adopt formal way to analyze and find all of the ineffective test cases.

3. Obtaining a pair-wise relationship set of input parameters

3.1 The dependent relationships among input parameters

Schroeder and Korel (Schroeder & Korel, 2000) used Input-Output relationships to reduce combinatorial test cases. Based on their idea, many researchers (Saraph, et al., 2003) (Cheng, et al., 2003) offer a variety of methods to reduce the combinatorial test suite via Input-Output relationships. But they all do not consider the dependent relationships among input parameters.

For many PUTs, the dependent relationships among some values of one input parameter and the values of another parameter are sometimes able to be obtained and useful. In these cases, not all combinations of values of input parameters are meaningful or valid for combinatorial testing, and the dependent relationships among input parameters should be systematically consider during modeling and generating combinatorial test suite.

If we use the dependent relationships of input parameters to reduce the combinatorial test suite, we must analyze and obtain these dependent relationships firstly. For the sake of the convenience of the discussion, we present two hypotheses as follows.

The dependent relationships of input parameters are defined and there is an existing set of dependent relationships for a PUT.

The dependent relationships of input parameters are one way and have transmissibility, but have no circulation.

There are also some dependent relationships between two parameters or among more than two parameters. There are some kinds of dependent relationship as follows (Silberschatz, et al., 2005):

One-one(1:1): The subset or all of values of one parameter dependent on the subset or all of values of another parameter, such as B dependents on A in Figure 1.

One-many(1:*n*): The subset or all of values of one parameter dependent on the subsets or all of values of many other parameters, such as D and E dependents on C in Figure 1.

Many-one(*n*:1): The subsets or all of values of many parameters dependent on the subset or all of values of one parameter, such as D dependents on B and C in Figure 1.

Many-many(*m*:*n*): The subsets or all of values of many parameters dependent on the subsets or all of values of other more than one parameter.

In these four types of relationships, 1:1 type of relationship is called two-parameter dependence relationship, which we can use the relationships of this type directly. The implication of 1:n type of relationship is one depended parameters decide n depending parameters, which sometimes we may transform some relationships of 1:n type into n relationships of 1:1 type and then use them. Whereas, n:1 and m:n type of relationships can not be used to obtain the pair-wise combinations of input parameters, but can be used to reduce the test suite.

We use the more succinct mark form to label the dependent relationships.

(a:b) represents parameter value b depends on parameter a(1:1 type).

($a \wedge b$:c) represents parameter value c depends on two parameter values a and b(n:1 type).

(a:$b \vee c$) represents parameter values b or c depends on parameter value a (1:n type). We may transform it into (a:b) and (a:c).

(a:$b \wedge c$) represents parameter values b and c depends on parameter value a (1:n type).

($a \wedge b$:$c \vee d$) represents parameter values c or d depends on parameter values a and b (m:n type). We may transform it into ($a \wedge b$:c) and ($a \wedge b$:d).

($a \wedge b$:$c \wedge d$) represents parameter values c and d depends on parameter values a and b (m:n type).

For example, date includes year, *month* and *day*. Suppose the values of *year* belong to {1999, 2000}; the values of *month* belong to {1, 2, …, 12}; the values of *day* belong to {1, 2, …, 31}. We analyze the dependent relationships for several special dates. The dependent relationships among *year*, *month* and *day* are as follows:

(1) The *day* value 29 in Feb. depends on *year* value 2000 and *month* value 2.

(2) The *day* value 30 depends on *month* value 1,3,4,5,6,7,8,9,10,11,12.

(3) The *day* value 31 depends on *month* value 1,3,5,7,8,10,12.

So we have the dependent set of input parameters as fellows.

(1){2000:2∧29}.

(2){1∨3∨4∨…∨12:30} or {1:30, 3:30, 4:30, …, 12:30}.

(3){1∨3∨5∨7∨8∨10∨12:31} or {1:31, 3:31, 5:31, 7:31, 8:31, 10:31, 12:31}.

In (2) and (3), the elements in dependent sets of input parameters are all two parameters dependent relationships. The element value in pair-wise combinations must be in accordance with the dependent relationships.

3.2 Obtaining input parameters relationships

This is one of the main problems for our method to gather input parameters relationships. There are many ways of gathering input parameters relationships. We can analyze program documentation (e.g., program specifications, or a cause-effect graph, should it exist) for the relationships and we also can analyze program codes to obtain the relationships. This is called as static analysis. Another way of analysis is dynamic, which analyze returning information during program execution so as to obtain some information we need.

The static analysis of program code includes program constraint code or glue code among components. For example, there is a program code which has date input parameters. There are several situations as fellows.

(1)Non any constrains among date input parameters in program codes. Then we can not obtain any information on relationships among input parameters.

(2)Having a piece code of constrains, seeing Figure 2(a). We can obtain some information on relationships among input parameters via analyzing the code.

(3)Having some output statements in program codes, seeing Figure 2(b). We can obtain some information on relationships among input parameters via analyzing the returning or output information during program execution.

We can analyze these program documentation or program codes via the manual way. Due to the diversification of form of program documentation, the manual way is only able to be adopted. But the way is complex and low efficiency. There already are automated techniques of analysis program codes to help us obtain the information of input-output parameter relationships. But automated analysis of input parameter relationships is difficult since codes sometimes are too complex. Whereas the method of analyzing returning or output information has both advantages and limitations. We hardly obtain all input relationships in a program. We had better adopt all kinds of methods to obtain more information of input parameters relationships as possible.

4. Reducing pair-wise combinatorial test suite by using the dependent relationships

After we obtain the pair-wise dependent relationship set, we can reduce pair-wise combinatorial test suite using dependent relationship set. Since existing methods of generating pair-wise combinatorial test suite are mature, we may use them to reduce test suite. There are two main ways to use the set of dependent relationship of input parameters.

(1)Remove the ineffective pair-wise combinations from existing set of pair-wise combinations, then cover the set without ineffective pair-wise combinations to generate test suite.

(2)Modify the existing methods so that examine a test case according to the set of dependent relationship of input parameters and judge whether the test case is a ineffective test case before put it into the pair-wise combinatorial test suite.

For the algorithms of PIO and AETG, the first way is easy to obtain test suite, since the PIO and AETG need a precondition i.e. the covered pair-wise combination set π. So we may remove the ineffective pair-wise combinations in advance and this do easily. The second way is more complex relatively.

In order to attain our aim which is to obtain a effective test suite, we firstly give the algorithm by which determine whether a given test case is invalid test cases. See the Figure 3.

In this section, we have a detailed account of reducing process of pair-wise combinatorial test suite using dependent relationships of input parameters. But, what demands notice is that if a PUT has ineffective *k*-wise combinations, we will also remove them from test suite.

5. Experiments

In our experiments, there are mainly two parts. One is the implementation of generating a pair-wise combination set without ineffective combinations. Another is generate the pair-wise combinatorial test suite which covers the pair-wise combination set.

The empirical study in this paper gives the results of several applied cases. All tests are performed on PC with a 2.2 GHz Intel CPU and 512 Mb of memory. Computer O.S. is Windows XP and the program code is edited, compiled and run on C++ Builder 6. Before the program running, an initialized text file must be built in advance, in which includes the number of input parameters, the symbolic value sets for each input parameters and the set of dependent relationships of pair-wise input parameters.

In the empirical study, there are six are practical applied cases at all. Details of the six cases used in the empirical study, and we can see the empirical results in Table II. Table II mainly shows the time which is taken in removing these ineffective 2-way combinations from the set of 2-way combinations. Although the parameters numbers of our subjects are not large, it is acceptable in consumption of time.

From Table II we can see the comparison of generating pair-wise test cases between two algorithms, and between before and after reducing ineffective 2-way combinations of input parameters. In general, the number of after reducing ineffective 2-way combinations is less than the number of before reducing ineffective 2-way combinations.

In Table II and Table III, the meaning of *Subject* Col. is that m^n: n-the number of input parameters, m-the number of input parameter values; $m^n * q^p$: $n+p$-the number of input parameters; and the number of n input parameters is m and the number of p input parameters is q.

6. Conclusions

This paper presents a method to reduce the pair-wise combinatorial test suite and remove the ineffective test cases from test suite so as to make the testing of PUTs smoothly. The method is based on analysis the dependent relationships among input parameters. The main thing in the method is how to obtain ineffective pair-wise combinations of input parameters. In addition, how to modify these existing methods is a key too. Experiments data tell us our method will have good effects in the number of test cases.

Although the method we propose reduces pair-wise combinatorial test suite, but still has some limitations for some PUTs. Future research work mainly focuses on how to obtain the pair-wise dependent relationships of input parameters from program documents or codes.

References

Cheng C., Dumitrescu A. and Schroeder P. (2003). Generating Small Combinatorial Test Suites to Cover Input-Output Relationships. *Proceedings of the Third International Conference on Quality Software (QSIC'03)*, 76-82.

Cohen D. M., et al. (1997). The AETG system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*. 23(7), 437-444.

Czerwonka J. (2009). Pairwise Testing. [Online] Available: http://www.pairwise.org/tools.asp.

Mandl R. (1985). Orthogonal Latin squares: An application of experimental design to compiler testing. *Communications of the ACM*, 28(10), 1054-1058.

Saraph P., Last M., & Kandel A. (2003). Test Case Generation and Reduction by Automated Input-Output Analysis. *Proceedings of 2003 IEEE International Conference on Systems, Man & Cybernetics (SMC 2003),* Washington, D.C., USA, 5–8.

Schroeder P. J. & Korel B. (2000). Black-Box Test Reduction Using Input-Output Analysis. *Proceedings of the 2000 ACM SIGSOFT International Symposium on Software testing and Analysis*, Portland, Oregon, United States, 173-177.

Shiba T., Tsuchiya T., & Kikuno T. (2004). Using artificial life techniques to generate test cases for combinatorial testing. *Proceedings of the 28th International Computer Software and Applications Conference (COMPSAC'2004)*, Hong Kong, China, 72-78.

Silberschatz A., Korth F.H. and Sudarshan S. (2005). *Database System Concepts (Fifth Edition),* New York, NY: The McGraw-Hill press.

Tai K. C., & Lei Y. (2002). A test generation strategy for 2-dimension testing. *IEEE Transaction Software Engineering*, 28 (1), 109-111.

Table 1. Coverage of pair-wise tests for sort command

	Height	Width	Avoids	Cases	Block(%)	Decision(%)	C-uses(%)	P-uses(5)
A	10	13	0	126	92	80	75	72
B	4	23	0	41	95	86	77	75
Avg.					93.5	83	76	73.5

Table 2. These results of algorithm of removing ineffective combinations

No.	Subject	Num. of 2-way combinations	Num. of ineffective 2-way combinations	time (s)
1	4^3	54	5	<0. 1
2	4^4	96	7	<0. 1
3	5^4	160	15	<0. 1
4	5^5	250	28	<0. 1
5	6^4	240	27	<0. 1
6	$3^3*4^4*2^5$	148	7	<0. 1

Table 3. The comparison of generating test cases between two algorithms, and between before and after reducing ineffective combinations

No.	Subject	Number of pair-wise test cases (PIO)		Number of pair-wise test cases(AETG)	
		Before reducing	After reducing	Before reducing	After reducing
1	4^3	9	9	9	9
2	4^4	10	10	10	10
3	5^4	11	11	10	10
4	5^5	14	13	12	11
5	6^4	12	11	11	10
6	$3^3*4^4*2^5$	11	10	10	10

$$\begin{tikzcd}
A \arrow[r] & B \arrow[r] \arrow[rr, bend left] & C \arrow[r] \arrow[rr, bend right] & D & E
\end{tikzcd}$$

Figure 1. Dependent relationship illustration

```
if(month==2){
if(year%4==0&&(year%100!= 0||year%400== 0))
{
day = 29;
}
else{
day = 28
}
}
else
if (month==4||month==6||month==9||month==11){
day = 30;
}
else{
day = 31;
}
```

(a)

```
if(month==2){
if(year%4==0&&(year%100!= 0||year%400== 0))
{ day = 29;
System.out.println(month+ "month value is :1-"+day);
}else{
day = 28;
System.out.println(month+ "month value is :1-"+day);
}
} else
if (month==4||month==6||month==9||month==11){
day = 30;
System.out.println(month+ "month value is :1-"+day);
}else{
day = 31;
System.out.println(month+ "month value is :1-"+day);
}
```

(b)

Figure 2. Pieces of program codes

```
YNUselessTestCase(dependtR,t) //dependtR is the set of dependent relationships and t is a test case
{   $r_i$=1;dependtRNum;//the element number of dependent relationship set;
    lb1=0;lb2=0;//
    while ($r_i$<= dependtRNum)
    {   Obtain No. $r_i$ dependent relationships in set of dependtR;
        Obtain the depending parameter value $v_k$ in No. $r_i$ dependent relationships;
        Obtain $v_k$ subscript value k ;
        Obtain the No. k element value $tv_k$;
        lb1= compare($tv_k$,$v_k$);
        If（lb1=0）{$r_i$=$r_i$+1;}
        Else
        {   n=Obtain the number of depended parameters in No. $r_i$ dependent relationships;
            A=Obtain a vector which is made of the depended parameters in No. $r_i$ dependent
                relationships;
            B=Obtain a vector whose elements is corresponding with A in test case t in subscript
                value;
            lb2=compareV(n,A,B);
            If（lb2=0）//
            {   removing test case t from test suite;
                ri=ri+1;}
          }}
}
int compare(x,y) // It is a comparing function.
{   if(x==y) then return 1;
    else return 0;
}
int compareV(n,A,B) // It is a vector comparing function.
{   if(A==B) then return 1;
    else return 0;
}
```

Figure 3. The determining algorithm of ineffective test case