Volume 3, Issue 8, August 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Implementation of Test Case Prioritization Technique Using Static Path

Yogita Garg, Arun P. Agrawal

Deptt. of Computer Science & Amity University, Noida

India

Abstract- Test case prioritization is an efficient and realistic technique for regression testing. It is helpful in enhancing the effectiveness of regression testing by categorizing and executing test cases according to their significance. Regression testing is applied to modified software to ensure that the modified parts behave as desired and the unmodified parts have not been negatively affected. A test suite shall be reused in regression testing which may have recurring and redundant test cases. In order to reduce the cost of maintaining the test suite and implementing the test cases in regression testing, we prioritize those test cases that can test & track partial changes. In this paper, we implement a technique for test case prioritization in which test cases can be prioritized on the basis of static function call path. According to the Zhang Zhi-hua's algorithm, we can find a change point with the minimum set of changes in function for regression testing, and show it with the help of an example. The result shows that, the algorithms can significantly reduce the size and the cost of the test suite for regression testing, and achieves good cost effectiveness.

Keywords- Test case prioritization; Function call; Static Path; Change point; MTP; MPC;

I. Introduction

Regression testing is a testing method to ensure software quality in the software development process. As soon as the software enters into the maintenance phase after its deployment, the accuracy of the software can be defoliated due to the changes introduced. During regression testing, reusing previous test cases on the basis of selection and prioritization can improve the efficiency and reduce the cost of software testing. Also it can resolve issues of lack of experience of testers. Considering the different test cases with different importance, a priority is assigned to each test case. Test cases are selected and rerun in accordance with their priority order, and this prioritization method could achieve higher efficiency of error detection[3]. As a kind of regression testing technique, to achieving this task we adopt a technique of static path coverage. This is a kind of heuristic algorithms, is used to predict which test cases should be first executed in the testing. In this paper, we show an implementation of a technique – function call path using C language. We applied this algorithm on an example of Insurance department.

II. Related Term

Test case prioritization (TCP) is the problem of ordering the test cases within the test suite of a system under test (SUT), with the goal of maximizing some criteria, such as the fault detection rate of the test cases (Wong et al. 1997). Should the execution of the test suite be interrupted or stopped for any reason, the more important test cases (with respect to the criteria) have been executed first. More formally, Rothermel et al. (2001) define the TCP problem as follows.

A. *Definition 1 (Test case prioritization):*

Given: T, a test suite; PT, the set of permutations of T; and f , a performance function from PT to the real numbers.
Find: $T` \in PT$ s.t. $(\forall T`` \in PT)(T`` \neq T`)[f(T`) \geq f(T``)]$.

In this definition, PT is the set of all possible prioritizations of T and f is any function that determines the performance of a given prioritization. The definition of performance can vary, as developers will have different goals at different times (Rothermel et al. 2001)[1]. Developers may first wish to find as many faults as possible; they may later wish to achieve maximal code coverage. In these scenarios, the problem definition of TCP is the same, but the performance function f being optimized changes. Researchers have proposed and evaluated many techniques to solve TCP problem, based on a range of data sources and prioritization algorithms.

B. *Definition 2 (Static Path):*

The Static Path described in this paper is defined as sequence from entry point to end point of program. Thinking about the control logic of software, we analyze source code and get all the executable call sequence, which is called static path.
For example:

Pseudo-code for the Insurance Premium Program

```
Dim driverAge, points as Integer
Dim baseRate, premium As Real

1.  Input(baseRate, driverAge, points)
2.  Premium = 0
3.  Select case DriverAge
4.  Case 1: 16<=driverAge<20
5.          ageMultiplier = 2.8
6.         if points<1 Then
7.              safeDrivingReduction =50
8.  End If
9.  Case 2 : 20<=driverAge < 25
10.       ageMultiplier = 1.8
11.      If points < 3 Then
12.           safeDriverReduction = 50
13.      End If
14. Case 3 : 25<=driverAge<45
15.      ageMultiplier = 1 #
16.      If points < 5 Then
17.        safeDrivingReduction =100
18.      End If
19. Case 4 : 45<=driverAge< 60
20.      ageMultiplier = 0.8
21.      If points < 7 Then
22.           safeDrivingReduction  = 150
23.    End If
24. Case 5: 60<= driverAge < 120
25.      ageMultiplier = 1.5
26.      If points < 5 Then
27.            safeDrivingReduction = 200
28.   End If
29. Case 6 : Else
30.            Output("Driving Age out of range")
31. End Select
32. Premium = baseRate * ageMultiplier – safeDrivingReduction
33. Output(premium)
```

1, 2, 3 ……..32, 33 are sequence no of each step. Different static path is such as

Table1- Static Path for Insurance Program Premium

Path	Node Sequence
P1	1-2-3-4-5-6-8-31-32-33
P2	1-2-3-4-5-6-7-8-31-32-33
P3	1-2-3-9-10-11-13-31-32-33
P4	1-2-3-9-10-11-12-13-31-32-33
P5	1-2-3-14-15-16-18-31-32-33
P6	1-2-3-14-15-16-17-18-31-32-33
P7	1-2-3-19-20-21-22-23-31-32-33
P8	1-2-3-19-20-21-22-23-31-32-33
P9	1-2-3-24-25-26-28-31-32-33
P10	1-2-3-24-25-26-27-28-31-32-33
P11	1-2-3-29-30-31-32-33

As a kind of regression testing technique for improving the efficiency of test case, heuristic algorithms are used to predict which test cases should be first executed in the testing[4].

C. Definition 3 (Change point):

Software developers modify (or add, delete) source code to correct defects, we consider that the condition which the change own to is changed, in the static path the modified path is called change point.

D. Definition 4 (Matrix for test case covering path):

It is a Boolean matrix expressed as MTP= $(k_{i,j})_{n \times m}$, used to record correspondence between testing cases and function call path, element $k_{i,j} = 1$ if and only if test t_i coverage function call path j, otherwise, $k_{i,j} = 0$. The construction of matrix is relatively simple, corresponding elements are set according to the relationship between test cases and function call path. If there are *n* test cases in test suite, and function call paths of software are *m*, time complexity of the construction algorithm is less than *O (nm)[6]*.

E. Definition 5 (Matrix for paths and change points):

It is a Boolean matrix expressed as MPC($k_{i,j}$)$_{m \times p}$, used to record the relation between each function call path and change points, elements $k_{i,j} = 1$ if and only if path r_i contains change point j, or $k_{i,j} = 0$. The construction of MRM (matrix)[7] is also simple, traverse each function call path, set the value of corresponding element according to it includes the change point or not. If there are m function call paths, change points are p, time complexity of the algorithm is less than O(mp).

III. Algorithm Description:

Using matrix MTP, we obtain the number of different change points which covered by each test case, regard this number as the test case priority. Finally, test cases will be sorted in descending order according to priority. Based on the above analysis, the prioritization algorithm PNC[8] (Prioritizing by Numbers of Changes) could be obtained (As shown in algorithm).

A. Algorithm

Input: set of all test cases $T=\{t_1,t_2 \ldots t_n\}$, MTP,MPC, weight α about number of change point, weight 1-α about occurrence times of change points[2]
Output: set of test cases after sorting T_c

```
/* initialization* /
for i=1 to n // for each test case, the same priority
    ti.pri=0; // is assigned, such as 0
    ti.mod= ∅; //set of change point is initialized to ∅
endfor
for i=1 to m //Initialization
     ri.mod= ∅;
     ri.times=0;
endfor
for i=1 to m
     for j=1 to p // sum up for each path
        if MPC[i,j]=1 then
           ri.mod= ∪j cj; //different change point
           ri.times++; // occurrence times of change point
        endif
    endfor
endfor
for i=1 to n
    for j=1 to m //sum up for each test case
        if MTP[i,j]=1 then
            ti.mod= ∪j rj.mod; //different change point
            ti.pri+= rj.times; // occurrence times
        endif
   endfor
   ti.pri= ti.pri*(1-α); // occurrence times has less weight
   ti.pri= ti.pri +|ti.mod| *α // the other has greater weight
endfor
Sort(T); // descending sort by ti.pri
```

B. Implementation of Insurance Code in C language

```
#include <stdio.h>
#include <conio.h>
```



```
void main()
{
int points, drive_age,safeDrivingReduction;
float premium,base_rate,age_multiplier;
clrscr();
printf("\nPlease enter base rate, drive age, and points");
scanf("%f%d%d",&base_rate,&drive_age, &points);
premium = 0.0;
if (drive_age>=16 && drive_age<20)
        drive_age = 1;
else
        if(drive_age>=20 && drive_age<25)
        drive_age = 2;
else
        if(drive_age>=25 && drive_age<45)
        drive_age = 3;
else
        if(drive_age>=45 && drive_age<60)
        drive_age = 4;
else
        if(drive_age>=60 && drive_age<120)
        drive_age  = 5;

switch(drive_age)
{
  case 1:age_multiplier=2.8;
           if(points<1)
           {
             safeDrivingReduction = 50;
           }
          break;
case 2:   age_multiplier=1.8;
          if(points<3)
          {
            safeDrivingReduction = 50;
          }
          break;
case 3:   age_multiplier=2.0;
          printf("\nAge = %f", age_multiplier);
          if(points<5)
          {
            safeDrivingReduction = 100;
          }
          break;
case 4:age_multiplier=0.8;
          if(points<7)
          {
            safeDrivingReduction =150;
          }
          break;
case 5:   age_multiplier=1.5;
          if(points<1)
          {
           safeDrivingReduction = 200;
          }
          break;
default: printf("\n Drive age is out of range");

}
```



```
printf("\nbase rate = %f", base_rate);
printf("\nage_multiplier= %f", age_multiplier);
printf("\nsafeDrivingReduction = %d", safeDrivingReduction);
premium =base_rate * age_multiplier - safeDrivingReduction;
printf("\n premium = %f", premium);
getch();
}
```

C. Implementation of Zhang Zhi-hua's Algorithm in C Language

```
#include<stdio.h>
#include<conio.h>
void sort(int tpri[]);
void main()
{
   int i, j, tpri[10], tmod[10];
   int MTP[3][3], MPC[3][3];
   int rmod[10], rpri[10], tc[10], rtimes[10];
   int a;
   //Input value of matrix for test case covering path
   for (i=1;i<=3;i++)
      for(j=1;j<=3;j++)
           scanf("%d", &MTP[i][j]);
  //Input value of Matrix for path and change points
   for(i = 1;i<=3;i++)
      for(j=1;j<=3;j++)
           scanf("%d", &MPC[i][j]);
  for(i=1;i<=10;i++)
  {
    tpri[i] = 0;
    tmod[i] = 0;
  }
  for(i=1;i<=3;i++)
  {
    rmod[i]=0;
    rtimes[i]=0;
 }
for(i=1;i<=3;i++)
          for(j=1;j<=3;j++)
          {
              if( MPC[i][j] == 1)
             {
             rmod[i] = 1;
               rtimes[i]++;
             }
          }
          for (i=1;i<=10;i++)
          {
             for(j=1;j<=3;j++)
             {
               if (MTP[i][j]== 1)
               {
                     tmod[i]= rmod[j];
                     tpri[i]+=rtimes[j];
               }
           }
             printf("value of a = %d", a);
             tpri[i] = tpri[i] * (1-a);
             tpri[i] =tpri[i] + tmod[j] * a;
         }
```



```
        sort (tpri);
        getch();
}
void sort(int tpri[10])
{
 int i, j, temp, k;
 printf("Insertion sort\n");
 printf ("\n Array before sorting :\n");
 for (i=0;i<=10;i++)
   printf("%d\t", tpri[i]);
   for (i=1 ; i<=10;i++)
        for( j=0; j<i;j++)
        {
         if(tpri[j]>tpri[i])
           {
              temp = tpri[j];
              tpri[j]=tpri[i];
              for(k = i;k>j;k--)
                    tpri[k] = tpri[k-1];
              tpri[k+1] = temp;
           }
        }
 printf("\n Array after sorting: \n");
 for(i=0;i<=10;i++)
  {
     printf("%d\t", tpri[i]);
  }
}
```

D. Valid Input

Let's suppose we have seven different test cases via T1, T2, T3, T4, T5, T6, T7. Initial Priority of these test cases are T1 : 0, T2 : 0, T3 : 0, T4 : 0, T5 : 0, T6 : 0, T7: 0. We have eleven different path as shown in table1. Such as P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11. Lets suppose change points are C1, C2, C3. .MTP[i,j] It is a Boolean matrix expressed as MTP= $(k_{i,j})_{n \times m}$, used to record correspondence between testing cases and function call path, element $k_{i,j} = 1$ if and only if test t_i coverage function call path j, otherwise, $k_{i,j} = 0$.This matrix contains seven test cases and eleven static Paths. So the matrix MTP[7,11] is in Figure 1

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
T1	0	0	0	1	1	1	0	0	0	0	0
T2	1	0	0	0	0	0	0	0	0	0	1
T3	1	0	0	0	0	1	0	1	1	1	0
T4	0	1	1	0	1	1	0	0	0	0	0
T5	0	1	0	0	0	0	1	0	0	0	0
T6	1	1	1	0	1	0	1	0	0	0	0
T7	0	0	1	1	0	0	1	1	1	1	1

Figure1. A MTP[7,11] matrix for insurance premium program

Now MPC[i,j] It is a Boolean matrix expressed as MPC($k_{i,j}$)$_{m \times p}$, used to record the relation between each function call path and change points, elements $k_{i,j} = 1$ if and only if path r_i contains change point j, or $k_{i,j} = 0$. This matrix is of 11 static Path and 3 change points. So MTP[11,3] is in Figure 2 .

	C1	C2	C3
P1	1	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	1
P5	0	0	0
P6	0	0	0
P7	0	0	0
P8	0	0	0
P9	0	1	0
P10	0	0	0
P11	0	0	0

Figure2. A MPC[11,3] matrix for insurance premium program

IV. Result and Experimental Output

Where t_{pri} is a array of test case priority. According to the input, the experimental results are on implemented code , test cases are prioritize like T1:1; T2:1; T3:2; T4:0; T5:0; T6 :1; T7: 2;In this result T1, T2 and T6 gets one priority, T3 and T7 gets two priority. And T4, T5 gets lowest priority. So those test cases get higher number which tests are execute first rathet than lower test cases. The results on this input data are T3 > T7 > T1 > T2 > T6 > T4 > T5.

V. Algorithm Analysis:

After analysis it shows that time complexity is O(m*n)+O(n*p), where m is the number of test cases, n is the number of paths, p is the number of change points. Generally, m>p, so the algorithms time complexity is O(m*n). In addition, during the testing, if some paths which including change points are not covered, test cases need to be added, we can adjust their priorities based on the design information at this time. This implementation is complete and effective, and had been validated Zhang Zhi-hua's algorithm[5].

VI. Conclusion

It is an important research field that how to select test cases, reduce the cost of regression testing, and improve test efficiency .The set of prioritization algorithms proposed in this paper is a new exploration for regression testing prioritization technique which oriented function call path[7]. Think about testing historical factor have an impact on the priority of test case, expand the statement-level path coverage to function call-level path coverage, different priority is given to different test case according to the change points coverage of function call path, test cases are selected based on priority while regression testing[9]. Compared with existing algorithms, this set of prioritization algorithms improve the efficiency of regression testing and guarantee testing adequacy, because only the modified and affected parts of software are tested.

References

[1] Rothermel G,Untch R H,Chu C Y,et al. Prioritizing test cases for regression testing[J]. IEEE Transactions on Software Engineering, 2001,27(10), pp.929-948.

[2] ZHANG Zhi-hua MU Yong-min TIAN Ying-ai, Test Case Prioritization for Regression Testing Based on Function Call Path, Fourth International Conference on Computational and Information Sciences, 2012, pp.1372-1375

[3] Stephen W. Thomas, Hadi Hemmati, Ahmed E. Hassan, Dorothea Blostein Static test case prioritization using topic models. © Springer Science+Business Media, LLC 2012

[4] M J Harrold,J J Jones,T Li,et al. Regression test selection for Java software. In Proceedings of OOPSLA'O1.Tampa Bay, Florida, USA, 2001, pp.312-326.

[5] Wong W E,Horgan J R,London S,et al. A study of effective regression testing in practice[C].Proceedings of the8th IEEE International Symposium on Software Reli-ability Engineering. 1997, pp.264-274 .

[6] Cem Kaner. Improving the Maintainability of Automated Test Suites[EB /OL]. http: / /www. kaner. com / lawst1. htm, 1997.

[7] ZHANG Zhi-hua, MU Yong-min. Research of Path Coverage Generation Techniques Based Function Call Graph[J] . Acta Electronica Sinica ,2010 , 38 (8): pp.1808-1811 (in Chinese).

[8] Elbaum S,Malishevsky A G,Rothermel G. Prioritizing test cases for regression testing[C] //Proceedings of the International Symposium on Software Testing and Analysis,2000, pp.102-112.

[9] Srivastava A,Thiagarajan J. Effectively prioritizing tests in development environment[C] //Proceedings of the International Symposium on Software Testing and Analysis, 2002, pp.97-106.

[10] Jeffrey D,Gupta N. Test case prioritization using relevant slices[C]//Proceedings of Computer Software and Applications Conference, 2006, pp.411-420.

[11] Jones J, Harrold M J. Test-suite reduction and prioritization for modified condition/decision coverage[C]//Proceedings of the International Conference on Software Maintenance, 2001, pp.92-101.

[12] Qu Bo,Nie Changhai,Xu Baowen. Test case prioritization based on test suite design information [J].Chinese Journal of Computers, 2008, 31 (3) : pp.431-439 (in Chinese).

[13] Yu JIANG, Yongmin MU, Zhihua ZHANG. Modificatory Indentification Algorithm Research for Source Code Oriented[C]. The Second International Workshop on Education Technology and Computer Science, WuHan,China, 2010, Vol III: pp.388-391

[14] Zhiying JIANG, Yongmin MU, Zhihua ZHANG. Program Flow Graph Oriented Analysis of Coverage and Time Performance. 2010 2nd IEEE International Conference on Information Management and Engineering, ChengDu, China. 2010, Vol : pp.439-443

[15] ZHENG Yuhui, MU Yongmin, Zhihua HANG. Research on the Static Function Call Path generating Automatically. 2010 2nd IEEE International Conference on Information Management and Engineering, ChengDu, China. 2010, Vol : pp.405-409.

[16] Bing JIANG, Yongmin MU, Zhihua ZHANG. Research of Optimization Algorithm for Path-Based Regression Testing Suit [C].The Second International Workshop on Education Technology and Computer Science, WuHan,China, 2010, Vol II: pp.303-306

