International Journal of Computer Applications (0975 – 8887)
Volume 89 – No 16, March 2014

Fault based Test Suite Prioritization based on Minimal MUMCUT Strategy

Usha Badhera
Assistant Proffesor
Banasthali Vidyapith
Jaipur, India

Annu Maheshwari
M.tech Scholar
Banasthali Vidyapith
Jaipur, India

ABSTRACT

Logic expressions are widely used in specifications and in programs. Testing criteria which covers logic expressions implies a high probability of detecting faults. Fault-based test suite prioritization of test cases has been considered in this study.Test cases are generated from logic expressions in irredundant normal form(IDNF) derived from specifications or source code by applying Minimal-MUMCUT. The proposed approach directly utilizes the theoretical knowledge of fault-detecting ability of test cases. The effectiveness of prioritization techniques has been validated by an empirical study done on bench mark expressions using two different metrics APFD, and FATE.

Keywords

MUMCUT, MUTP, MNFP, CUTPNFP, Fault detection, APFD, FATE.

1. INTRODUCTION

Boolean expressions are central aspect of programs and specifications. Given n variables in expression 2^n distinct testcases are required for exhaustive testing .This is expensive even when n is modestly large.The possible solution is to select a small subset of all possible test cases which can effectively detect most of the common faults.

In the past decade, may testing criteria have been proposed for software characterized by complex logical decisions, such as those in safety-critical software[1],[2].[3]. In such cases, it is imperative that the logical decisions be adequately tested for the occurrence of plausible faults. In recent years, more sophisticated coverage criteria have been advocated, like BOR (Boolean OpeRator Testing Strategy),BMIS(Basic Meaningful Impact Strategy),modified condition/decision coverage (MC/DC) [1],[2],[4] and the MUMCUT criteria[5].Compliance of the MC/DC criterion has been mandated by Federal Aviation Administration for the approval of airborne software. MC/DC was ‘‘developed to address the concerns of testing Boolean expressions and can be used to guide the selection of test cases at all levels of specification, from initial requirements to source code’’ [1]. While MC/DC is effective in fault detection, it also consumes a great deal of testing resources [2],[6].

MUMCUT strategy is to generate test cases that can guarantee detection of seven types of single faults provided that the original expression is in irredundant disjunctive normal form(IDNF)[6]. In this strategy, there is no restriction on the number and occurance of variables in the given Boolean expressions. Minimal-MUMCUT [7] that improves the MUMCUT strategy by considering the feasibility problem of the three testing constituents of the MUMCUT strategy. It reduces the test suite size as compared to MUMCUT without compromising any fault detection capability. Thus, the extra tests required by the MUMCUT criterion are of little, if any, value based on the theoretical and empirical studies conducted [7].

In this study test cases generated by Minimal-MUMCUT are prioritized according to rate of fault detection and an algorithm is proposed for the same.The effectiveness of proposed approach is validated by APFD by comparing it with baseline approach.

2. PREMILINARY AND PREVIOUS WORK

This section covers the basics of previous work, to introduce the notation and terminologies to be used in this paper, as well as the main ideas that motivated this work. Section 2.1 presents Minimal-MUMCUT strategy for generating test cases from logical expressions . In Section 2.2, seven different types of single faults generated from irredundant disjunctive normal form(IDNF) of original expression and hierarchy of fault classes supported by Minimal-MUMCUT test cases for specification-based testing is presented. Test case prioritization technique and well-known metric APFD and FATE for evaluating the effectiveness of prioritized test cases is outlined in Section 2.3.

2.1 Minimal MUMCUT Strategy

The amalgamation of MUTP, MNFP and CUTPNFP strategies is referred to as the MUMCUT strategy[6]:

MUTP: A test set T is said to satisfy the MUTP strategy (or simply the U strategy) if, for every i, T contains UTPs of the ith term p_i of S such that all possible truth values (that is, 0 and 1) of every variable not occurring in p_i are covered. Such a test set T is called a Multiple Unique True Point (MUTP) test set.

MNFP: A test set T is said to satisfy the MNFP strategy (or simply the N strategy) if, for every i and j, T contains NFPs of the jth literal of the ith term p_i so that all possible truth values (that is, 0 and 1) of every variable not occurring in p_i are covered. Such a test set T is called a Multiple Near False Point (MNFP) test set.

CUTPNFP: A test set T is said to satisfy the CUTPNFP strategy (or simply the C strategy) if, for every i and j, as far as possible, T contains a UTP $\vec{t}$ of the ith term p_i and a NFP $\vec{f}$ of the jth literal of p_i such that $\vec{t}$ and $\vec{f}$ differ only in the corresponding truth value of the jth literal of p_i. Such a test set T is called a Corresponding Unique True Point Near False Point (CUTPNFP) test set.

According to Minimal-MUMCUT, if MUTP criterion is infeasible,then prioritzed test set will be MUTP(U) test cases followed by overlapping NFPs.Overlapping NFPs is a set covering combinatorial optimize test cases. For example : (a&b)|(b&!c)|(!b&c) MUTPs for this Boolean expression are : 111,010,001,101. In this expression (a&b) and (b&!c)terms are MUTP infeasible and only(!b&c) term is MUTP feasible which covers both values 0 and 1 for missing literal a.The NFPs for the expression (a&b)|(b&!c)|(!b&c) 011,101,000,010,100 but the overlapping NFPs are 100(b of a&b and c of !b&c),000(b of b&!c),011(a of a&b and b of !b&c and c of b&!c).

MUTP criterion is infeasible: Test Set = MUTP + NFP

If CUTPNFP criterion is infeasible then prioritized test set will be MUTP test cases followed by CUTPNFP. For example: (a&b)|(b&!c)|(!b&c) CUTPNFP criterion is infeasible for this Boolean expression. CUTPNFPs test cases for above Boolean expression are 111,011,100,010,000,001.

CUTPNFP criterion is infeasible :Test Set = MUTP+CUTPNFP

If MUTP criteria is feasible then prioritized test set will be MUTP(U) test cases followed by CUTPNFP(C) followed by MNFP(N) test cases.

MUTP feasible:Test Set =MUTP+PCUTPNFP+MNFP

Table 1 shows total number of test cases generated by Minimal MUMCUT,feasibility criteria of MUTP and CUTPNFP and total number of single fauts generated of seven types mentioned in Section 2.2

Table 1. Criteria Feasibility for Boolean Expressions

S. N	Predicate	Test Case	faults detect -ed	MUTP Criteria	CUTPN-FP Criteria
1	(a&!b&d)|(a&!c&d)| e)	9	93	Not Feasible	Feasible
2	(a&b)|(a&c)|(b&c)	6	56	Not Feasible	Feasible
3	(a&b&c)|(d&e)	7	68	Feasible	Feasible
4	(a&b)|(b&!c)|(!b&c)	7	53	Not Feasible	Not Feasible
5	(!a&b)|(c&d)	6	46	Feasible	Feasible

2.2. Fault Hierarchy

A fault is an error in the original Boolean expression.A faulty implementation is referred to as single-fault expression if (1) it differs from the original expression by one syntactic change; and (2) it is not equivalent to the original expression. This study considers the following classes of simple faults for logical decisions. A decision S in n variables can always be written in *disjunctive normal form* (DNF) as a sum of product.

Consider the following Boolean expression for example:

$$S = ab + \overline{c}d + e$$

Table 2. Types of Faults

Fault	Description	Example
Expression Negation Fault (ENF)	The expression or sub-expression is negated	$\overline{ab + \overline{c}d}$ + e
Term Negation Fault (TNF)	A term is negated	$\overline{ab} + \overline{c}d + e$
Term Omission Fault (TOF)	A term is omitted	$\overline{cd} + e$
Operator Reference Fault (ORF)	An OR operator(+) is implemented as the AND operator or vice versa	$ab.\overline{c}d + e$ or $a + b + \overline{c}d + e$
Literal Negation Fault (LNF)	A literal is negated	$\overline{a}b + \overline{c}d + e$
Literal Omission Fault (LOF)	A literal is omitted	$b + \overline{c}d + e$
Literal Insertion Fault (LIF)	A literal is inserted	$abc + \overline{c}d + e$
Literal Reference Fault (LRF)	A literal is implemented as another literal	$\overline{ac} + \overline{c}d + e$

Lau and Yu's Fault Hierarchy modified is displayed in Fig.1 based on how criterion feasibility affects fault detection as indicated by Kaminski and Ammann [7]. A solid arrow from a source fault to a destination fault indicates that if a test detects a source fault, it also detects a corresponding destination fault. When the MUTP criterion is infeasible, a test set detecting all LIFs is not guaranteed to detect all LRFs. Thus the solid arrow between the LIF and LRF in Lau and Yu's hierarchy is changed to a dashed arrow. In Lau and Yu's hierarchy no arrow exists between the LRF and LOF. A dashed arrow is added to represent that when guaranteeing detection of all LIFs does not guarantee detection of all LRFs (due to MUTP infeasibility), adding tests to detect the undetected LRFs will detect all corresponding LOFs (unless the PCUTPNFP criterion is infeasible). The reason is that when the MUTP criterion is infeasible but the PCUTPNFP criterion is feasible, a UTP will not detect an LRF but a corresponding NFP will. Since the Minimal-MUMCUT criterion always requires MUTP tests, the LIF is guaranteed to be detected. Since the Minimal-MUMCUT requires (1) PCUTPNFP tests when the MUTP criterion is infeasible and the PCUTPNFP criterion is feasible and (2) MNFP tests when the PCUTPNFP criterion is infeasible, LRF detection is guaranteed.

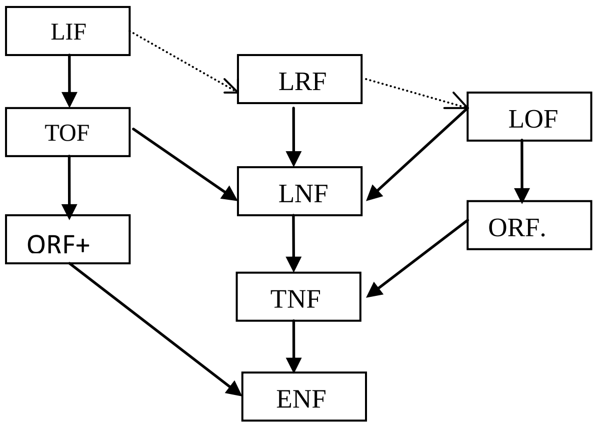

Fig.1: Fault Hierarchy Based on Infeasibility[7]

A MUTP test set guarantees to detect single faults of the classes

ENF, TNF, TOF, ORF(+), LNF and LIF [6],[9].

A MNFP test set guarantees to detect single faults of the classes

ENF, TNF, ORF(.), LNF and LOF [6],[9].

Chen and Lau [6] proved that, when a CUTPNFP test set is combined with a MUTP and a MNFP test set, the resulting test set will always detect all the faults in Table1.

2.3 Test Case Prioritization

Test case prioritization techniques schedule test cases in an execution order according to some criterion.Test case prioritization problem is defined[8] as follows:

Given: T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.

Problem: Find T' belongs to PT such that (for all T") (T" belongs to PT) (T" ≠ T') [f (T') ≥ f(T")].

Here, PT represents the set of all possible prioritizations (orderings) of T and f is a function that, applied to any such ordering, yields an award value for that ordering.

The performance of the prioritization technique used is known as effectiveness It is necessary to assess effectiveness of the ordering of the test suite. Effectiveness will be measured by the rate of faults detected. The following metrics are used to calculate the level of effectiveness:

2.3.1 Average Percentage Of Faults Detected (Apfd) Metric

APFD (Average Percentage Fault Detected) metric is a measure of how rapidly a prioritized test suite detects faults, which measures the weighted average of percentage of faults detected over the life of a test suite. [13] ,[8].The APFD used in this paper is calculated by taking the weighted average of the number of faults detected during the run of the test suite. APFD can be calculated using the following notations:

Let T - The test suite under evaluation

m - the number of faults contained in the program under test P

n - The total number of test cases and

TFi - The position of the first test in T that exposes fault i.

$$\text{APFD} = 1 - \frac{TF1+TF2+TF3+TF4+TF5 \ldots\ldots\ldots TFi}{m*n} + \frac{1}{2*n}$$

APFD can be calculated when prior knowledge of faults is available. APFD values ranges from 0 to 100; higher value implies faster (better) fault detection rates.

2.3.2 Fault Adequate Test Set Size (Fate) Metric

A different effectiveness metric, called Fault-Adequate Test set sizE (FATE) is defined as the size of a minimal fault-adequate subset of the prioritized test suite[11]. For a fault-adequate test suite of size *n*, the FATE value ranges between 0 and *n*.A lower FATE value is preferred as it implies earlier detection of all target faults. For a fair comparison of test suites of different sizes, the value of FATE may be normalized by expressing it as a percentage of the size of the whole (unprioritized) fault-adequate test suite. A normalized FATE (abbreviated as *n*FATE) value always ranges between 0% and 100%.

2.3.3 Prioritization using Mumcut

By definition, a MUMCUT test set consists of points satisfying

the three component strategies: MUTP (U), MNFP (N) and CUTPNFP (C).These points are reffered as U-points, Npoints and C-points, respectively. The U, N and C strategies guarantee to detect faults of different classes, this knowledge has been employed to prioritizing a MUMCUT test cases and it has empirically proved that out of six combinations of U,N and C that are CNU,CUN,NCU,NUC,UCN,UNC, the UCN order gives better rate of fault detection[15].

3. PROPOSED WORK

In the proposed paper for prioritizing the test cases generated following steps are followed:

1. Single faults of seven types mentioned in Section 2.2, are generated using JAVA eclipse and JAVA collection framework.

2. For the given expression Minimal MUMCUT test cases are generated and feasibility criteria is tested.

3. Test cases are arranged according to the algorithm given in Section 3.1

4. The effectiveness of the prioritized test suite is assessed by calculating APFD and comparing it with Baseline approach.

3.1 Algorithm For Fault Based Prioritization Of Minimal Mumcut Tests

Input: Test suite T and number of faults detected by a test case

Output: Prioritized Test suite T'.

```
1. Begin
2. Set T' empty
3.  for each term X do
4.      If MUTP criteria is infeasible for X
            Prioritize Multiple Unique True Points (U)
            followed by overlapping NFPs(N)
5.          for each literal x in term X
6.              If CUTPNFP criteria is feasible for x
```

```
            **Prioritize** Unique True Points (U)
            followed by CUTPNFP(C)
7.      **end for**
8.  **else**
        **Prioritize** Mutiple Unique True Points (U) followed
        by overlapping NFPs(N) followed by CUTPNFPs
9.  **end for**
10. **End**
```

4. EXPERIMENTS AND RESULTS

Tables 3(a),(b),(c),(d) shows the number of faults detected by a test case in the test suite for the Boolean expression (a&b&c)|(d&e). Fault F17 is referred as equivalent fault as it is generating same original expression after faulty implementation.

Table 3(a). Fault Detected by the test cases for Predicate (a&b&c)|(d&e).

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21
t1			×		×						×		×		×						×
t2			×		×						×		×			×				×	
t3			×		×							×		×				×			
t4			×		×						×	×		×					×		
t5	×	×		×	×	×			×		×	×									
t6	×			×	×			×		×	×	×	×								
t7	×	×		×	×		×			×	×	×	×	×							

Table 3(b). Fault Detected by the test cases for Predicate (a&b&c)|(d&e).

	F22	F23	F24	F25	F26	F27	F28	F29	F30	F31	F32	F33	F34	F35	F36	F37	F38	F39	F40
t1				×	×	×					×				×				×
t2				×	×	×						×				×			
t3			×				×	×											
t4	×	×					×	×											
t5				×			×		×	×									
t6						×		×									×	×	×
t7					×			×					×	×	×				

Table 3(c). Fault Detected by the test cases for Predicate (a&b&c)|(d&e).

	F41	F42	F43	F44	F45	F46	F47	F48	F49	F50	F51	F52	F53	F54	F55	F56	F57	F58	F59
t1										×	×		×	×	×		×	×	×
t2	×										×	×		×	×	×		×	×
t3		×	×			×	×												
t4		×		×				×											
t5			×	×	×							×							
t6						×	×		×										
t7						×		×	×								×		

Table 3(d). Fault Detected by the test cases for Predicate (a&b&c)|(d&e).

	F60	F61	F62	F63	F64	F65	F66	F67	F68	F69
t1		×								
t2	×									
t3					×	×	×		×	×
t4			×	×		×	×	×		×
t5			×							
t6									×	
t7								×		

For the (a&b&c)|(d&e) predicate MUTP criteria is feasible for all terms and CUTPNFP criteria is feasible for all literals.

m= no. of faults = 68

n = no. of test cases = 7

So putting the values of m , n ,TFi(The position of the first test in T that exposes fault i) in the following equation :

$$APFD = 1- \frac{TF1+TF2+TF3+TF4+TF5 \ldots\ldots\ldots TFi}{m*n} + \frac{1}{2*n}$$

$$APFD = 1- \frac{5+5+1+5+1+5+7+6+5..............................3+3}{7*68} + \frac{1}{2*7}$$

APFD = 0.641

APFD = 64.1%

The value of **FATE metric is 7** since it is required to execute all the test cases to detect all the faults.Similarly Table 4 shows values of metrics APFD and FATE for some number of Boolean expressions.

Table 4. Values of APFD and FATE

S N .	PREDICATE	APFD of Proposed Appraoch	APFD of Baseline Appraoch	FATE
1	(a&!b&d)|(a&!c&d)| e)	67.63%	54.5%	9
2	(a&b)|(a&c)|(b&c)	56.33%	57.4%	6
3	(a&b&c)|(d&e)	64.1%	58%	7
4	(a&b)|(b&!c)|(!b&c)	58.1%	58%	7
5	(!a&b) |(c&d)	61.3%	60%	6

Table 4 shows the comparison between APFD values obtained from proposed approach and baseline approach. The proposed approach gives better Average Percentage of FaultsDetected during life time of a test suite.

4.1 Comparison Of Proposed Approach With Baseline

The comparison is drawn between APFD value of Boolean expression (a&b&c)|(d&e) using UCN order and baseline approach, which shows that value of APFD obtained using UCN order is more than baseline approach. The baseline approach uses concept of binary number to arrange test cases in serial order. For Boolean expression (a&b&c)|(d&e) the order of test cases is:

t3-00111- (7 in binary)

t5-01101-(13)

t7-10110-(22)

t4-11011-(27)

t6-11010-(28)

t1-11101-(29)

t2-11110-(30)

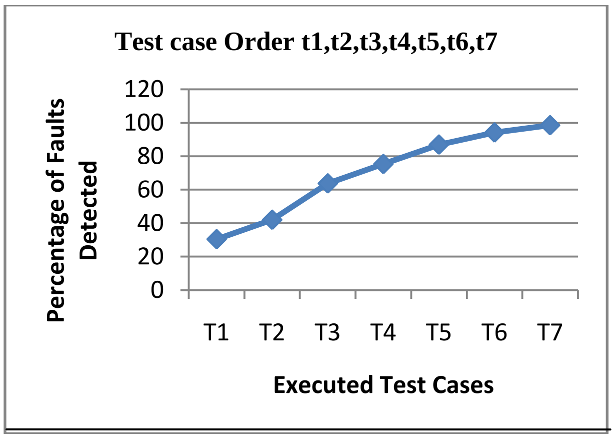

Fig 3:Graph for Boolean expression (a&b&c)|(d&e) using UTPs followed by Overlappinf NFPs Order with 64.1% APFD

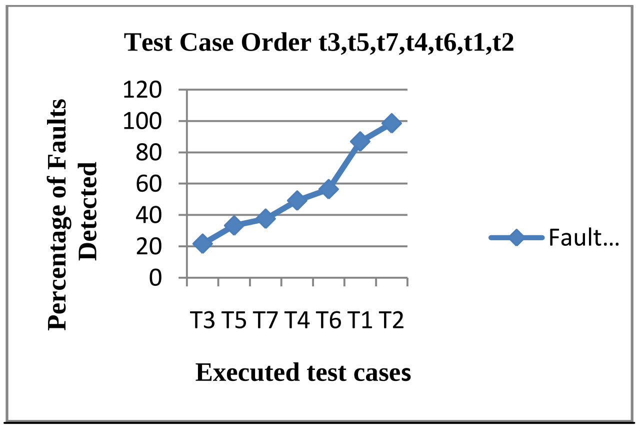

Fig 4 : Graph for Boolean expression (a&b&c)|(d&e) using BaseLine Technique with 58% APFD

The proposed order of prioritized test cases yield a higher average APFD where MUTP criteria is feasible than those prioritized with baseline approach. A normalized FATE value for all prioritized test cases is 100% ,FATE metric shows that all test cases of Minimal-MUMCUT are required to detect all eight classes of single faults.

5. CONCLUSION & FUTURE WORK

This paper proposed an algorithm for Prioritization of Minimal MUMCUT test cases in order to improve regression testing.In proposed study the experiments were done on some expressions where MUTP feasible expressions provide higher value of APFD metric. In this study effectiveness has been done for prioritized test cases with the help of APFD(Average Percentage Fault Detection) and FATE(Fault Adequate test set sizE) metrics.However,it is guaranteed that if even a single test is removed from a Minimal-MUMCUT test set, fault detection will be sacrificed for the fault types in Lau and Yu’s fault hierarchy[6].In future the experiment need to be conducted on the Boolean expression having more no of literals and order of prioritization need to be validated for high rate of fault detection.

6. REFERENCES

[1] Chilenski, J.J., Miller, S.P., 1994. Applicability of modified condition/decision coverage to software testing. Software Engineering Journal 9 (5), 193–229.

[2] Dupuy, A., Leveson, N., 2000. An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software. In: Proceedings of Digital Aviation Systems Conference (DASC 2000).

[3] Chilenski, J.J., 2001. An investigation of three forms of the modified condition decision coverage (MCDC) criterion. Tech. Rep. DOT/ FAA/AR-01/18, Federal Aviation Administration, US Department of Transportation, Washington, DC.

[4] Jones, J.A., Harrold, M.J., 2003. Test-suite reduction and prioritization for modified condition/decision coverage. IEEE Transactions on Software Engineering 29 (3), 195–209.

[5] Yu Y.T.,Lau M.F., Chen T.Y.,2005 "Automatic generation of test cases from Boolean specifications using the MUMCUT strategy" Journal of Systems and Software 79(6), 820–840.

[6] Lau M.F., Chen T.Y,2001 "Test Case Selection strategies based on Boolean Specifications" Software Testing, Verification and Reliability,11(3), 165-180

[7] Kaminski, G., & Ammann, P., 2009, "Using a fault hierarchy to improve the efficiency of DNF logic mutation testing" In *Software Testing Verification and Validation,ICST'09. International Conference on* (pp. 386-395). IEEE

[8] Elbaum S.,Malishevsky A.G.,Rothermel G.,2002,"Test case prioritization: a family of empirical studies", IEEE Transactions on Software Engineering 28 (2) , 159–182.

[9] Yu Y.T.,Lau M.F., 2006,A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions, Journal of Systems and Software 79 (5),577–590.

[10] Chen Z.Y., Fang C.R.,XU B.W.,2012," Comparing logic Coverage Criteria on test Case Prioritization".

[11] Yu Y.T., Lau M.F., 2012, "Fault based Test Suite Prioritizaton for Specification based Testing", Information and Software technology 54, 179-202

[12] Malishevsky A.G., Rothermel G. and Elbaum S. ,2002,"Modeling the Cost-Benefits Tradeoffs for Regression Testing Techniques" Proceedings of the International Conference on Software Maintenance (ICSM'02).

[13] Malishevsky A. G.,Ruthruff J. R., Rothermel G. ,Elbaum S. ,2006, "Costcognizant Test Case Prioritization".

[14] Elbaum S.,Rothermel G., Kanduri S.,Malishevsky A.G.,2004, "Selecting a Cost-Effective Test Case Prioritization Technique".

[15] Balance A. W., Vilkomir S., Jenkins W., 2012, "Effectiveness of Pair-wise Testing for Software with Boolean Inputs", IEEE Fifth International Conference on Software Tesing, Verification and Validation.

[16] Yu Y.T.,Lau M.F., 2002 "Prioritization of test cases in MUMCUT test sets: an empirical study",Proceedings of International Conference on Reliable Software Technologies, pp. 245–256.