SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Reliab. 2017; **00**:1–23
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/stvr

Prioritizing MCDC Test Cases by Spectral Analysis of Boolean Functions

T. Ayav

Izmir Institute of Technology, Dept. of Computer Engineering, 35430, Urla-Izmir, Turkey

SUMMARY

Test case prioritization aims at scheduling test cases in an order that improves some performance goal. One performance goal is a measure of how quickly faults are detected. Such prioritization can be performed by exploiting the Fault Exposing Potential (FEP) parameters associated to the test cases. FEP is usually approximated by mutation analysis under certain fault assumptions. Although this technique is effective, it could be relatively expensive compared to the other prioritization techniques. This study proposes a cost-effective FEP approximation for prioritizing Modified Condition Decision Coverage (MCDC) test cases. A strict negative correlation between the FEP of a MCDC test case and the influence value of the associated input condition allows to order the test cases easily without the need of an extensive mutation analysis. The method is entirely based on mathematics and it provides useful insight into how spectral analysis of Boolean functions can benefit software testing. Copyright © 2017 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Test prioritization; MCDC; spectral analysis; Boolean functions; mutation analysis

1. INTRODUCTION

Reports show that the cost of testing can be more than the half of total cost of software development process [1]. In regression testing and even in initial testing, running all of the test cases can require very large amount of effort. It is stated that in a typical software having 20,000 lines of code may require seven weeks to run the entire test suite [2]. In this case, a tester may want to order the test cases according to some objective so that those with the highest priority are run earlier.

Test case prioritization allows to schedule the execution of test cases in an order that attempts to maximize some objective function. The objective is usually about how quickly faults are detected in testing process. An improved rate of fault detection can provide a faster feedback about the program under test and let software engineers begin locating and correcting of faults earlier. Like several other studies in the literature [2, 3], this work focuses on increasing the likelihood of revealing faults earlier in testing process.

Several studies state that the FEP prioritization is one of the most effective techniques [3, 4, 5]. The disadvantage however is the difficulty in computing the FEP, which is generally based on an expensive mutation analysis. This work proposes a novel approach to estimating FEP values. It relies on the spectral analysis of Boolean functions, particularly the influence values of the input conditions. In the places where Boolean expressions are used, such as the requirements or the structure of the software, one can benefit from many useful and powerful techniques like Boolean derivative calculus and spectral analysis of Boolean functions. Boolean derivative calculus is particularly used in hardware testing [6, 7, 8, 9] and it is rarely exploited in software testing except for the MCDC test suite definition [10].

The proposed spectral analysis-based technique prioritizes MCDC test suites. MCDC is used in development of not necessarily the most critical software as per DO-178B standard. It is

Copyright © 2017 John Wiley & Sons, Ltd.

quite popular in practice. MCDC requires that each decision takes every possible outcome, each condition in a decision takes every possible outcome and each condition in a decision is shown to independently affect the outcome of the decision. As an example, consider the following trivial expression:

Expression = A and B

where A and B are conditions. To show the independence of A, condition B must be held to *true*; otherwise switching A will not affect the outcome of the expression. Therefore, the test pair corresponding to A will be (`T`,`T`) and (`F`,`T`). Independence of B is shown similarly and the test pair is found to be (`T`,`T`) and (`T`,`F`). Table I shows the three test cases.

Table I. MCDC test cases for expression A *and* B

A	B	A *and* B
T	T	T
F	T	F
T	F	F

The aim of the test prioritization is to change the order of tests such that tests with higher FEP can be applied earlier. FEP values of the individual tests are computed by the help of mutation analysis. The key idea of this work is to find a relationship between the fault exposing potentials and the influence values, which completely eliminates the mutation analysis and provides prioritization in a simple and fast manner compared to the existing methods.

The organization of the article is as follows: Section 2 explains MCDC criteria and formally how test cases are derived, presenting a demonstrative example. The effectiveness of the test suites and their fault coverages are computed by the mutation analysis, adopting some fault hypotheses and fault assumptions. These concepts are explained shortly in Section 3. Influence values are defined as a part of spectral analysis of Boolean functions in Section 4 with a thorough explanation of the spectral analysis. Section 5 presents the proposed prioritization technique of this work in detail. The relationship between the fault exposing potentials and the influence values is presented and explained using mathematics. The spectral analysis-based prioritization is shown step by step through a demonstrative example with the comparison to the FEP prioritization. The correctness of the prioritization is shown with a brute force approach by validating the method on a large number of Boolean functions in Section 6. The method is also demonstrated on a real system and compared with an existing prioritization algorithm from the literature. Section 7 discusses the potential threats to validity followed by the related work given in Section 8. Finally the article is concluded with Section 9.

2. MODIFIED CONDITION DECISION COVERAGE TESTING

The Modified Condition Decision Coverage (MCDC) is a control-flow testing criterion and it is required to verify Level A software as per the DO-178B standard [11]. MCDC provides a structural coverage analysis. Structural coverage criteria usually include i) Statement ii) Decision iii) Condition iv) Condition/Decision v) MCDC and vi) Multiple Condition in the order from the weakest to strongest criteria. Statement coverage ensures that every executable statement in a program is invoked at least once. This coverage is relatively weak since it is insensitive to the control structures. Decision Coverage (DC) ensures that every decision has taken all possible outcomes at least once. For example, consider the following code excerpt:

```
if A or B and C then
        true_statement;
else
        false_statement;
```

Copyright © 2017 John Wiley & Sons, Ltd.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

```
end if;
```

The only test pair for DC is defined as follows:

$$TP = \{(a, b) : f(a) = f'(b)\} \tag{1}$$

Two test cases where A=F, B=F, C=F and A=T, B=F, C=F satisfy the requirement of DC. However, the effects of B and C are not tested. For a compound condition, if two or more combinations of components of the condition could cause a particular branch to be executed, decision coverage will be complete when just one of the combinations has been tested. Compound conditions are a frequent source of code bugs, therefore DC can be considered as a weak criterion.

Condition coverage requires that every condition in a decision takes on all possible outcomes at least once. The previous problem is now settled yet it does not require the decision evaluated to both true and false. For example the test cases where A=F, B=T, C=T and A=T, B=F, C=F satisfy the requirement of condition coverage but not decision coverage since both test cases yield to true. Condition/decision coverage, as the name implies, combines condition and decision coverages. For example, the test cases where A=F, B=T, C=F and A=T, B=F, C=T satisfy the requirement of condition/decision coverage.

Multiple condition requires that each possible input combination constitutes a test case. This coverage is the strongest one and reveals all faults but it can be impractical as the number of inputs increases. For a decision with n inputs, multiple condition coverage requires 2^n test cases.

MCDC states that every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken all possible outcomes at least once, every decision has taken all possible outcomes at least once, and each condition in a decision has been shown to independently affect that decision's outcome by varying just that condition while holding all other possible conditions fixed. There are several forms of MCDC yet *masking* and *unique-cause* are the most common ones [12]. In this study, the proposed method is demonstrated through the unique cause MCDC but the technique can be easily adapted to the other forms as well. For any decision $f(x)$, the test pair for condition x_i can be defined as [10]:

$$\begin{aligned} TP_i = \{(a, b) : \forall j \in \{1, \ldots, n\} : i \neq j, a_j = b_j, a_i = b_i', \\ f(a) = f'(b), \frac{\partial f}{\partial x_i}(a) = \frac{\partial f}{\partial x_i}(b) = \mathtt{T}\} \end{aligned} \tag{2}$$

where $\frac{\partial f}{\partial x_i}$ is the derivative of f with respect to x_i and it is given with the following formula [7]:

$$\frac{\partial f}{\partial x_i} := f(x_i \leftarrow \mathtt{F}) \text{ xor } f(x_i \leftarrow \mathtt{T}). \tag{3}$$

Note that the derivative is *true* if switching x_i changes the function output and *false* otherwise. The above formal definition of test pairs should be read as:

- a and b are two test vectors that form MCDC independence pair for the ith condition.
- Condition i must toggle between two tests ($a_i = b_i'$)
- All bits of a and b except for i are equivalent ($\forall j \in \{1, \ldots, n\} : i \neq j, a_j = b_j$).
- Expression must return different results for the two tests ($f(a) = f'(b)$)
- Condition i must have an influence on the outcome of the expression when the first test is applied ($\frac{\partial f}{\partial x_i}(a) = \mathtt{T}$)
- Condition i must have an influence on the outcome of the expression when the second test is applied ($\frac{\partial f}{\partial x_i}(b) = \mathtt{T}$)

Unique cause MCDC wants the conditions of no interest to be fixed in the independence pair. Therefore the maximum number of pairs would be the number of conditions. On the other hand, masking MCDC allows any number of conditions to change so long as only the condition of interest has influence on the outcome of the expression. This generally results in more tests and it is stronger than unique cause MCDC. In theory, minimum $n + 1$ and maximum $2n$ test cases are required by

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

unique cause MCDC [12]. Note that, in the rest of the article, the test pairs are arranged such that $f(a) = \mathtt{F}$ and $f(b) = \mathtt{T}$ for the sake of clarity. By MCDC, unique cause MCDC is meant. The following code excerpt taken from the book of Ammann and Offutt [13] will be used to explain MCDC:

```
if(((curTemp < dTemp-thresholdDiff)
    || (override && curTemp < overTemp - thresholdDiff))
    && (timeSinceLastRun > minLag) )
{...}
```

A variable or a literal in a Boolean expression represents a condition that cannot be further decomposed into simpler Boolean expressions. A condition may be either a simple Boolean variable representing the switch behaviour or it may actually represent a relational expression like "`curTemp < dTemp-thresholdDiff`". Therefore, assigning the following Boolean variables to the relational expressions, we have:
x_1: `curTemp < dTemp-thresholdDiff`
x_2: `override`
x_3: `curTemp < overTemp - thresholdDiff`
x_4: `timeSinceLastRun > minLag`
where $\mathbf{x} = [x_4, x_3, x_2, x_1]^T$ is the test vector. Therefore, this decision can be expressed with the following Boolean function:

$$f(\mathbf{x}) = (x_1 + (x_2 \cdot x_3)) \cdot x_4$$

Boolean derivatives with respect to the inputs can be computed as follows:

$$\frac{\partial f}{\partial x_1}(\mathbf{x}) = (x_2' + x_3')x_4 \quad (4)$$

$$\frac{\partial f}{\partial x_2}(\mathbf{x}) = x_1' x_3 x_4 \quad (5)$$

$$\frac{\partial f}{\partial x_3}(\mathbf{x}) = x_1' x_2 x_4 \quad (6)$$

$$\frac{\partial f}{\partial x_4}(\mathbf{x}) = x_1 + (x_2 x_3) \quad (7)$$

Taking into account the derivatives and the Formula (2), a test pair for input i is the solution to the Boolean satisfiability problem $\frac{\partial f}{\partial x_1}(\mathbf{x}) = \mathtt{T}$. For example, test inputs satisfying $\frac{\partial f}{\partial x_1}(\mathbf{x}) = (x_2' + x_3')x_4 = \mathtt{T}$ would be $[\mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{X}]$. The independence pair therefore consists of the vectors $\mathtt{a} = \mathtt{TFFF} = 8$ and $\mathtt{b} = \mathtt{TFFT} = 9$. Note also that $f(a) = \mathtt{F}$ and $f(b) = \mathtt{T}$. For input x_2, $x_1' x_3 x_4 = \mathtt{T}$ can be satisfied by $[\mathtt{T}, \mathtt{T}, \mathtt{X}, \mathtt{F}]^{\mathtt{T}} = (12, 14)$. Test pairs for x_3 and x_4 can be found similarly as $[\mathtt{T}, \mathtt{X}, \mathtt{T}, \mathtt{F}]^{\mathtt{T}} = (10, 14)$ and $[\mathtt{X}, \mathtt{F}, \mathtt{F}, \mathtt{T}]^{\mathtt{T}} = (1, 9)$ respectively. All test pairs are explicitly shown in Table II. The set of test pairs are then,

$$TP = \{(8, 9), (12, 14), (10, 14), (1, 9)\}.$$

Note that two test cases, `9` and `14` are repeating in two pairs. Elimination of the repeating test cases (test cases numbered with 6 and 8) reduces the size of the test suite to six. The resulting test suite is then,

$$T = \{8, 9, 12, 14, 10, 1\}.$$

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

Table II. MCDC test cases for expression $f(\mathbf{x}) = (x_1 + (x_2 \cdot x_3)) \cdot x_4$

Test No	x_4	x_3	x_2	x_1	$\mathbf{x}, f(\mathbf{x})$
1	T	F	F	**F**	8, F
2	T	F	F	**T**	9, T
3	T	T	**F**	F	12, F
4	T	T	**T**	T	14, T
5	T	**F**	T	F	10, F
6	T	**T**	T	F	14, T
7	**F**	F	F	T	1, F
8	**T**	F	F	T	9, T

According to MCDC, conditions that occur more than once in a decision need each occurrence to demonstrate its independence. This limits the applicability of MCDC to singular functions, where each condition has a single occurrence only. For example, consider the following non-singular function:

$$f(\mathbf{x}) = x_1(x_2 + x_3) + x_1' x_2'$$

The problem comes about when testing the first occurrence of x_1. By definition, the first occurrence of x_1 will be toggled between true and false, while holding the second occurrence of x_1 along with the other conditions fixed. This is not possible and there is no clear guidance how MCDC is applied to this type of Boolean expressions. Nevertheless, these problems have been mitigated with various methods [14] so that MCDC can still be applied to a large variety of functions. Moreover, the applicability of MCDC to non-singular functions is not a great concern since those functions are rarely seen in real systems [12].

MCDC is not limited to structural code coverage only. In general, it is an effective method to test Boolean functions where applicable, like branch testing, requirement testing or specification testing [15, 16]. For example, cause-effect graphing is a popular requirement-based testing introduced by Myers several decades ago [17, 18] and a recent study proposes the application of MCDC to generate test cases from these graphs [19]. For further information on MCDC analysis, there exist some fundamental studies [12, 14, 20], including a systematic literature overview [21].

3. FAULT CLASSES AND FAULT COVERAGES OF TESTS

Circuit testing is traditionally such that typical hardware manufacture defects are hypothesized and then test cases are derived to detect them. Similar approaches exist in software testing but the fact that the source of defects is human can introduce much broader range of faults, which might be harder to hypothesize. In fault-based testing or measurement of the fault-detection effectiveness, *mutation analysis* is a common approach [22]. Mutation analysis is a technique that makes small changes to the correct program under certain fault hypotheses so that test cases can be generated to find these faults. Mutation analysis is based on two assumptions: (1) the competent programmer hypothesis, which states that programmers tend to develop correct programs, and (2) the fault coupling effect, which hypothesizes that a test set that detects all simple faults in a program is also capable of detecting more complicated faults. Many testing techniques have been proposed to derive test cases based on Boolean expressions. Boolean expressions are found in logical predicates inside programs and specifications which model complex conditions. Assume a Boolean predicate P with n conditions/variables. For n conditions, it is possible to have $2^{2^n} - 1$ Boolean expressions that can be considered as faulty. Let them be denoted P'. The aim of testing is to distinguish P from the faulty rest of the expressions, that is $P \oplus P' = \texttt{TRUE}$. Multiple condition criterion requires that 2^n tests are sufficient for this purpose but for larger n's this becomes harder. When only $m < 2^n$ tests are selected, $2^n - m$ combinations are left uncovered. The fault coverage of m tests can be given as,

$$P_{(n,m)} = 1 - \frac{2^{(2^n - m)} - 1}{2^{2^n}}. \tag{8}$$

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

It is inefficient and unrealistic to consider all the possible combinations of Boolean expressions. The faulty versions generated by typical mistakes of programmers are taken into account instead. Several fault hypotheses are proposed for this purpose, as listed below [15, 23, 24, 25, 26]:

Variable Negation Fault (VNF) A variable is wrongly implemented as its negation. For example, $f(x)$ can be implemented as $(x_1' + (x_2 \cdot x_3)) \cdot x_4$.

Expression Negation Fault (ENF) The whole expression or its subexpression is wrongly implemented as its negation. For example, $f(x)$ can be implemented as $(x_1 + (x_2 \cdot x_3))' \cdot x_4$.

Missing Variable Fault (MVF) A condition is omitted in the expression. For example, f can be implemented as $(x_1 + (x_2 \cdot x_3))$.

Variable Reference Fault (VRF) A variable is replaced by another variable. For example, $f(x)$ can be implemented as $(x_1 + (x_2 \cdot x_3)) \cdot x_3$.

Operator Reference Fault (ORF) A Boolean operator is replaced by another. For example, $f(x)$ can be implemented as $(x_1 \cdot (x_2 \cdot x_3)) \cdot x_3$.

Stuck-at-Zero (SA0) A condition is stuck at false value. For example, SA0 fault of variable x_1 results in $(x_2 \cdot x_3) \cdot x_4$.

Stuck-at-One (SA1) A condition is stuck at true value. For example, SA1 fault of variable x_1 results in x_4.

Clause Conjunction Fault (CCF) Condition c_1 is replaced by $c_1 \wedge c_2$. For example, f can be implemented as $(x_1 + (x_2 \cdot x_3)) \cdot (x_4 \cdot x_1)$.

Clause Disjunction Fault (CDF) Condition c_1 is replaced by $c_1 \vee c_2$. For example, f can be implemented as $(x_1 + (x_2 \cdot x_3)) \cdot (x_4 + x_1)$.

Given a large specification and a large program, considering all possible fault classes may be quite expensive in terms of computational resource and time complexity, particularly when an optimal solution is of interest. One approach is to find hierarchical relationships between these classes so that stronger classes can be sufficiently used to reduce the size of the problem. A fault type, δ_1 is said to be stronger than fault type, δ_2 if any test guaranteed to detect all possible faulty implementations of type δ_1 will also detect those of δ_2. The relationship is denoted by $\delta_1 \geq_f \delta_2$.

The first effort was Kuhn's study [23] that theoretically establishes a hierarchy between VNF, ENF and VRF classes such that ENF $\geq_f$ VNF $\geq_f$ VRF. Tsuchiya and Kikuno extended Kuhns three fault classes to include the fault class of a missing condition [24]. Lau and Yu further extended Kuhns hierarchy by analyzing the relationships between variable faults and literal faults [25]. All of these studies restricted Boolean expressions into Disjunctive Normal Form (DNF). In fact, the expression do not have to appear in DNF and it has been shown that a single fault in a general expression may correspond to more than one fault in its corresponding DNF [26]. Kapoor and Bowen extended the analysis to general Boolean specifications introducing new fault classes [27]. Finally, Chen et. al. extended Kapoor's study, identifying the incorrect relationships and proposing a new fault class hierarchy, as shown in Figure 1 [28]. In conclusion, among nine fault classes, it is sufficient to take into account only four of them, ORF, CCF, CDF and ENF, which can considerably facilitates the mutation analysis. In this article, the proposed technique will be demonstrated using ORF, CCF, CDF and ENF fault classes under single fault assumption.

4. SPECTRAL ANALYSIS OF BOOLEAN FUNCTIONS

This section introduces the fundamental concepts of the spectral analysis of Boolean functions, which benefit the prioritization of test cases. Spectral or Fourier analysis is widely used in mathematics and engineering. Fourier decomposes a signal as a sum of periodic functions like

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

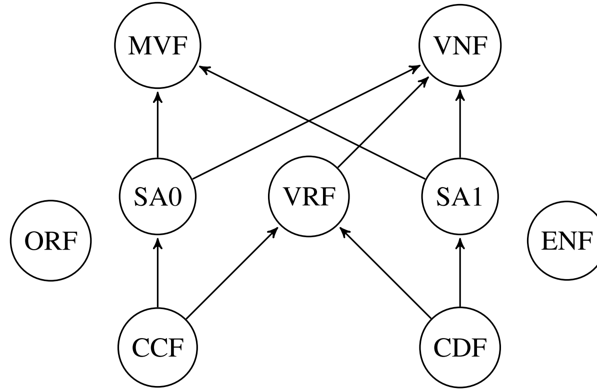

Figure 1. Fault class hierarchy [28].

$\chi_y(x) = e^{2\pi ixy/n}$. In case of Boolean functions, the most used transform has been defined over Abelian group $\mathbb{Z}_2^n$. Boolean functions are usually defined as $f : \{\mathtt{F}, \mathtt{T}\}^n \to \{\mathtt{F}, \mathtt{T}\}$. As a requirement of Fourier analysis, instead of 0 and 1, 1 and -1 will be used as *false* and *true* values respectively. Hence the function becomes $f : \{1, -1\}^n \to \{1, -1\}$. The relevant definitions and theorems about Fourier analysis are given below without the proofs. For further explanations, examples and theorems with proofs, refer to the seminal works of O'Donnell [29, 30] and Wolf [31].

Theorem 1
(Fourier expansion). Every $f : \{-1, 1\}^n \to \mathbb{R}$ can be expressed with its Fourier expansion,

$$f(x) = \sum_{\omega \subseteq [n]} \widehat{f}(\omega)\chi_\omega(x), \tag{9}$$

where $\widehat{f}(\omega)$ is the Fourier coefficient and $\chi_\omega(x) = \prod_{i\in\omega} x_i$ is the parity function. It is also adopted that $\chi_\emptyset \equiv 1$.

Definition 1
(Inner product). Let $f, g : \{-1, 1\}^n \to \{-1, 1\}$. The inner product between f and g is defined as

$$\langle f, g\rangle := \sum_{x\in\{-1,1\}^n} \frac{f(x)g(x)}{2^n} = \mathop{\mathbf{E}}_{x\in\{-1,1\}^n}[f(x)g(x)].$$

Note that $\langle f, f\rangle = \|f\|_2^2 = 1$ and more generally $\|f\|_p := \mathbf{E}[|f(x)|^p]^{1/p}$.

Fourier coefficients can be written as

$$\widehat{f}(\omega) = \langle f, \chi_\omega\rangle = \mathop{\mathbf{E}}_{x}[f(x)\chi_\omega(x)]. \tag{10}$$

Note in particular that coefficient $\widehat{f}(\emptyset) = \mathbf{E}[f]$ corresponds to the mean $\mathbf{E}[f]$. This parameter is also called the *balance* of the function. For example, recall the 3-input Boolean function $f(x) = a \vee b \wedge c$ that was derived in the previous section. Assuming that a and c are the most and least significant bits respectively, it is trivial to derive the truth vector for f as [F F F T T T T T], i.e., $[1\ 1\ 1\ -1\ -1\ -1\ -1\ -1]$ as shown in Table III. By using Formula (10), Fourier coefficients can be computed as $\widehat{f}(\emptyset) = \frac{1}{2^3}(1+1+1-1-1-1-1-1) = -0.250$, $\widehat{f}(1) = \frac{1}{2^3}(1\cdot 1 + 1\cdot 1 + 1\cdot 1 - 1\cdot 1 - 1\cdot -1 - 1\cdot -1 - 1\cdot -1 - 1\cdot -1) = 0.750$ and so on. The eight coefficients are used to constitute the Fourier expansion of f as follows:

$$f = -0.25 + 0.75a + 0.25b + 0.25ab + 0.25c + 0.25ac - 0.25bc - 0.25abc.$$

Note that for 3-input Boolean function $f(a, b, c)$ with a being the most significant bit and c being the least significant bit, the eight spectral components are shown in Figure 2. The Fourier coefficients shown in Figure 3 are the correlations between these components and the function output.

Copyright © 2017 John Wiley & Sons, Ltd.

Table III. Truth table for $f(x) = a \vee b \wedge c$.

x	a	b	c	$f \in \mathbb{B}$	$f \in \mathbb{R}$
0	F	F	F	F	1
1	F	F	T	F	1
2	F	T	F	F	1
3	F	T	T	T	-1
4	T	F	F	T	-1
5	T	F	T	T	-1
6	T	T	F	T	-1
7	T	T	T	T	-1

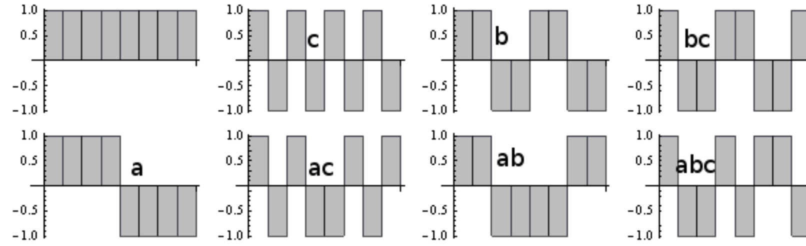

Figure 2. The eight spectral components of a 3-input Boolean function.

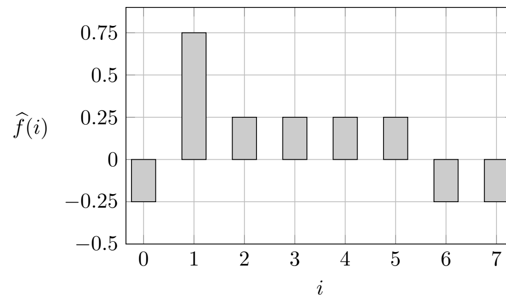

Figure 3. Fourier coefficients of $f(x) = a \vee b \wedge c$.

The derivative or difference calculus for Boolean functions has been exploited in testing digital circuits and also software over the past two decades. It can also be used to describe the notion of influence. The following definition of the derivative slightly differs from the one given with Equation 3 and better represents the classical notion of derivative:

Definition 2
(Derivative). The derivative of f with respect to its input x_i is defined as,

$$\frac{\partial f}{\partial x_i} = \frac{f(x_i \coloneqq -1) - f(x_i \coloneqq 1)}{(-1) - 1} \tag{11}$$

$$= \sum_{\omega \ni i} \widehat{f}(\omega) \chi_{\omega \setminus i}(x). \tag{12}$$

For example, $\frac{\partial f}{\partial a} = \frac{-1-(0.5+0.5b+0.5c-0.5bc)}{-2} = 0.75 + 0.25b + 0.25c - 0.25bc$. It can be noticed that this derivative would produce 0 if b and c are true ($b = c = -1$) and 1 otherwise. f is monotonic, i.e., non-decreasing since changing one bit from false to true would never cause the output to switch from true to false. Monotonicity requirement can also be expressed by $\frac{\partial f}{\partial x_i} \geq 0$, for all i. In theory, all Boolean functions excluding the negation operation are monotonic.

Equation (11) requires that the derivative of a Boolean function can be in $\{-1, 0, 1\}$. If $\frac{\partial f}{\partial x_i}(x) = \pm 1$, then x_i is said to be pivotal for f at x. If it is zero, then x_i has no influence on f at x. Hence, the

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

influence of input x_i on f is the expected value of being pivotal over x. The definition of influence is as follows:

Definition 3
(Influence). The influence of input x_i on f is defined as,

$$\mathrm{Inf}_i(f) = \Pr[f(x) \neq f(x^{\oplus i})] = \mathop{\mathbf{E}}_x \left[\frac{\partial f}{\partial x_i}(x)^2 \right] = \sum_{\omega \ni i} \widehat{f}(\omega)^2. \tag{13}$$

where $x^{\oplus i}$ is the string x with its i-th bit flipped.

The total influence can be computed as [30],

$$\mathrm{I}(f) = \sum_{i=1}^{n} \mathrm{Inf}_i(f). \tag{14}$$

For $f(x) = a \vee b \wedge c$, the influence values are found as $\mathrm{Inf}_a(f) = 0.75$ and $\mathrm{Inf}_b(f) = \mathrm{Inf}_c(f) = 0.25$. Considering the influence values, a is the most important input, whereas b and c have identical importance and they are less important with respect to a. For monotonic functions, $\frac{\partial f}{\partial x_i}$ is greater than or equal to zero and the influence is [30],

$$\mathrm{Inf}_i(f) = \mathop{\mathbf{E}}_x \left[\frac{\partial f}{\partial x_i} \right] = \widehat{\frac{\partial f}{\partial x_i}}(\emptyset) = \widehat{f}(\{i\}). \tag{15}$$

Equation (15) implies that the influence of a variable equals to a single Fourier coefficient. This makes the computation of influences feasible particularly for large functions. For the previous example, $\mathrm{Inf}_a(f) = \widehat{f}(1)$, $\mathrm{Inf}_b(f) = \widehat{f}(2)$, and $\mathrm{Inf}_c(f) = \widehat{f}(4)$.

Another concept is the energy spectrum that may provide useful information about the noise sensitivity of a function.

Definition 4
(Energy spectrum). For any real-valued function $f : \mathbb{B}^n \to \mathbb{R}$, the energy spectrum $\mathbf{E}_f$ is defined by

$$E_f(k) := \sum_{|\omega|=k} \widehat{f}(\omega)^2 \qquad \forall k : 1 \leq k \leq n \tag{16}$$

where $|\omega|$ depicts the number of 1 bits in ω.

$E_f(\emptyset)$ is known as DC component, which is the zero-frequency component. If most of the Fourier mass is localized on high frequencies, then the function is sensitive to small perturbations, i.e., condition failures as shown in a sample spectrum given in the right panel of Figure 4.

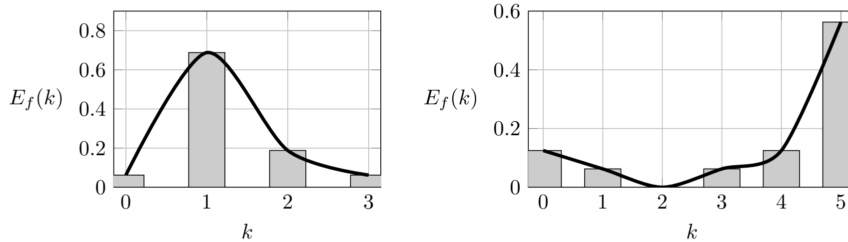

Figure 4. Energy spectrum of $f(x) = a \vee b \wedge c$ (Left Panel) and $f(x) = a \vee b \oplus c \oplus d \oplus e \oplus f$ (Right Panel).

From the testing point of view, it is easier to detect the faults of a Boolean expression that has a higher noise stability. This will be briefly demonstrated in the beginning of Section 5. Another useful parameter to analyze an expression is *Noise Stability*. If we define a fault probability for every single condition and denote it with ϵ, then the noise stability of a Boolean function can be defined as follows.

Copyright © 2017 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

Definition 5
(Noise stability). The noise stability of f at $1-2\epsilon$ is

$$Stab_{1-2\epsilon}(f) = \sum_{k=0}^{n}(1-2\epsilon)^k E_f(k). \tag{17}$$

The energy spectrum of the sample function $f(x) = a \vee b \wedge c$ is shown on the left panel of Figure 4. If the fault probability of a condition, ϵ, is assigned 0.25, the noise stability of the function can be calculated as 0.460938 for $\epsilon = 0.25$.

The influence for noisy inputs can also be defined as follows [30]:

Definition 6
(Noisy influence). The noisy influence of f at $1-2\epsilon$ is

$$\mathrm{Inf}_i^{1-2\epsilon}(f) = Stab_{1-2\epsilon}(\frac{\partial f}{\partial x_i}) = \sum_{\omega \ni i}(1-2\epsilon)^{|\omega|-1}\widehat{f}(\omega)^2 \tag{18}$$

4.1. Approximating Fourier Coefficients

The time and resource usage complexity of the transformation are given as $\mathcal{O}(n2^n)$ and $\mathcal{O}(2^n)$ respectively [32]. Therefore, transformation becomes harder as n gets bigger. In this case, Fourier coefficients can be approximated. Recall that the Fourier coefficient,

$$\widehat{f}(\omega) = \mathbb{E}[f \cdot \chi_\omega]$$

is an expectation under uniform distribution. $\chi_\omega : \{0,1\}^n \to \pm 1$ is a parity function defined as,

$$\chi_\omega(x) = (-1)^{\omega \cdot x},$$

where $\omega \cdot x = \sum_{i=1}^{n} \omega_i x_i = \sum_{i \in \omega} x_i$. We can approximate the Fourier coefficients from uniformly drawn examples $(x_1, f(x_1)), \ldots, (x_m, f(x_m))$. Expected value is the following empirical average,

$$\frac{1}{m}\sum_{j=1}^{m} f(x_j)\chi_\omega(x_j)$$

and this value converges to the exact value of $\widehat{f}(\omega)$ as m grows. Moreover, Chernoff bound indicates how quickly this convergence happens [33].

5. SPECTRAL ANALYSIS-BASED TEST CASE PRIORITIZATION

Spectral analysis provides useful information about the characteristics of Boolean functions, as explained in Section 4. The most common parameters are balance, influence, energy spectrum and noise stability. Many of these parameters are mutually dependent. For example, if zero-frequency component is high or Fourier mass is localized on left frequencies, the function is insensitive to small perturbations, hence its noise stability is expected to be high. $E_f(\emptyset)$ gives the balance value of a function in the range between -1 and 1. A zero or near-zero values mean that function is balanced. Unbalanced functions have also higher noise stability and lower influence values. Stability has positive correlation with balance values, whereas a negative one with total influence. Spectral analysis provides a sort of probabilistic reasoning, hence it gives useful information about the general characteristics of a function. For example, total influence or noise stability value may be exploited to reason about the effectiveness of a test suite generated for a Boolean function.

As an example, Table IV shows the results of an empirical study to demonstrate how stability is proportional to the average effectiveness of test suites generated by decision coverage criterion. In this empirical study, arbitrarily selected TCAS expressions numbered 1,4,9,19 and 20 are used

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

[34]. For each expression, all possible DC test suites are generated and average FEP values are calculated. It can be observed that $Stab/n$ is proportional to the FEP. In other words, the correlation between them is computed as 0.99. Note that stability values are computed for $1 - 2\epsilon = 0.6$, yet this correlation would be almost same for other ϵ values. Hence, decision coverage tests can be performed just by ordering the tests according to the stability values.

Table IV. FEP and Stability for DC test suite

TCAS #	1	4	9	19	20
n	8	5	7	8	7
$Stab_{0.6}(f)$	0.853084	0.54396	0.950812	0.746375	0.912888
$Stab_{0.6}(f)/n$	0.1066	0.10879	0.13583	0.09329	0.1304
$Average$ FEP	0.345	0.332	0.524	0.278	0.49

On the other hand, the influence value of an input condition, which is computed depending on the Fourier coefficients, gives an information about how much that condition influences the decision. Therefore, the influence is a suitable parameter to distinguish condition dependent test cases generated by condition coverage, condition/decision coverage or modified condition/decision coverage techniques.

This study focuses on the influence to distinguish MCDC test cases and prioritize them. The reason for selecting MCDC as the coverage technique is that it has been shown to be stronger than DC, CC and C/DC and it is a successful and widely adopted technique. Rothermel et. al. made the first formal definition for test prioritization [2]:

Definition 7
(Test Prioritization) Let T be a test suite; PT be the set of permutations of T and ν be a valuation function from PT to the real numbers.
Problem : Find $T' \in PT$ such that $\forall T'' \in PT$, $\nu(T') \geq \nu(T'')$.

Below, a key proposition that allows to prioritize MCDC tests with respect to the influence values is presented. A detailed explanation follows this proposition. As the main contribution of the paper, prioritization of MCDC tests is given with Definition 8. Finally, a simple example is conducted to demonstrate the entire approach.

Proposition 1
Let $f(x) : \{-1, 1\}^n \rightarrow \{-1, 1\}$ and $\mathrm{Inf}_i(f)$ be the influence of input x_i on f. Let $T = \{tp_1, tp_2, \ldots, tp_n\}$ be the MCDC test pairs and $\mathrm{FEP}(tp_i)$ be the fault exposing potential of tp_i under the assumption of ORF, CCF, CDF and ENF hypotheses. There exists a negative correlation between $\mathrm{Inf}_i(f)$ and $\mathrm{FEP}(tp_i)$.

This proposition sheds light on a promising relation between the influence and FEP, which facilitates the prioritization of the test cases. The intuition behind it is such that conditions that greatly influence decisions are unlikely to be tested by other test cases. However, conditions that barely influence decisions are highly likely to be tested by chance. As an example, Table V shows the truth vectors of $f(x) = a \vee b \wedge c$ and its three mutants under only VNF hypothesis for the sake of simplicity. The comparison of the two sample MCDC test pairs given below may help express the intuition. For example, pair $tp_1 = (0, 4)$ can be used to test condition a and pair $tp_2 = (1, 3)$ for condition b. It is expected that $f(0) = \mathtt{F}$, $f(4) = \mathtt{T}$ and similarly $f(1) = \mathtt{F}$, $f(3) = \mathtt{T}$ to pass the tests. In general, a test case with higher FEP value requires that the original function's output differs as greatly as possible from the most of the mutants' outputs. For most of the mutants, however, influence values remain almost the same. For instance, both $\mathrm{Inf}_a(f)$ and $\mathrm{Inf}_a(M_1)$ are equal to 0.75 and consequently a has a strong masking effect. Switching a in tp_1 from $\mathtt{F}$ to $\mathtt{T}$, switches also the output of M_2 and M_3 in the same direction, which means these test cases are unable to detect mutants M_2 and M_3. On the other hand, condition b has a less masking effect since $\mathrm{Inf}_b(f) = 0.25$ and pair tp_2 is indeed able to detect all the three mutants as can be clearly seen from the table.

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

Table V. Truth table for $f(x) = a \vee b \wedge c$ and its mutants $M_1(x) = a' \vee b \wedge c$, $M_2(x) = a \vee b' \wedge c$ and $M_3(x) = a \vee b \wedge c'$

x	a	b	c	f	M_1	M_2	M_3
0	F	F	F	F	T	F	F
1	F	F	T	F	T	T	F
2	F	T	F	F	T	F	T
3	F	T	T	T	T	F	F
4	T	F	F	T	F	T	T
5	T	F	T	T	F	T	T
6	T	T	F	T	F	T	T
7	T	T	T	T	T	T	T

Therefore, having a prioritization strategy that prioritizes test cases in the reverse order of their influence may help detect failures faster.

Let $T = \{tp_i^{p_i} \equiv (a_i, b_i)^{p_i} : \forall i \in \{1, 2, \ldots n\}\}$ be the test suite. $tp_i^{p_i}$ is such that priority p_i is associated to test pair tp_i. Priorities are adjusted as inversely proportional to the influences. Hence the valuation function should be designed as given in Equation (19).

$$\nu(T) = [\mathrm{Inf}_i(f) - \mathrm{Inf}_j(f)](i - j), \quad \text{where } i \neq j \text{ and } i, j \in \{1, 2, \ldots, n\}. \tag{19}$$

Therefore, test pairs are easily arranged in decreasing order by maximizing $\nu(T)$ in accordance with Definition 7. The proposed prioritization replaces Fault Exposing Potential (FEP) based prioritization. FEP of a test pair is proportional to its priority. FEP values are computed by the adoption of mutation analysis technique [35]. Mutation analysis creates a large number of faulty versions (mutants) of the expression by altering conditions and relations, and uses these to assess the quality of the test suites by measuring whether those suites can detect those faults (*kill* those mutants). The approach works as follows: Given a Boolean expression $f(x)$ and its test suite $T = \{tp_1, tp_2, \ldots, tp_n\}$, a set of mutants, $M_1, M_2, \cdots$, of f is created with respect to the chosen fault hypotheses. Next, each test case $tp_i \in T$ is checked against those mutants, noting whether tp_i kills them or not. The FEP of tp_i, $\mathrm{FEP}(tp_i)$ is the ratio of mutants of f killed by tp_i to the total number of mutants of f. Therefore, FEP can be expressed as follows:

$$\mathrm{FEP}(tp_i) := \Pr[tp_i = (a_i, b_i) \text{ detects } M_j = f(x^{\oplus j})]. \tag{20}$$

In other words, FEP values are explicitly computed using Formula (21).

$$\mathrm{FEP}(tp_i) := \frac{|M_j \text{ killed by } tp_i|}{|M_j|}. \tag{21}$$

Proof Sketch: Below, a proof sketch is given for Proposition 1. To compute the correlation $Corr(c_i, s_i)$, Pearson's formula given in Equation (22) is used.

$$Corr(c_i, s_i) = \frac{n \sum_{i=1}^{n} c_i s_i - \sum_{i=1}^{n} c_i \sum_{i=1}^{n} s_i}{\sqrt{\left(n \sum_{i=1}^{n} c_i^2 - (\sum_{i=1}^{n} c_i)^2\right)\left(n \sum_{i=1}^{n-1} s_i^2 - (\sum_{i=1}^{n-1} s_i)^2\right)}} \tag{22}$$

For the correlation to be negative, the following condition must be satisfied:

$$n \sum_i \mathrm{FEP}(tp_i)\,\mathrm{Inf}_i(f) - \sum_i \mathrm{FEP}(tp_i) \sum_i \mathrm{Inf}_i(f) < 0,$$

$$\mathrm{Inf}(f) \cdot \sum_i \mathrm{FEP}(tp_i) > n \sum_i \mathrm{FEP}(tp_i)\,\mathrm{Inf}_i(f)$$

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

It is clear that test pair $tp_i = (a_i, b_i)$ detects M_j if the following conditions are satisfied:

$$f(a_i^{\oplus j}) = 1, f(b_i^{\oplus j}) = -1, \tag{23}$$

$$\frac{\partial f}{\partial x_j}(a_i)^2 \cdot \frac{\partial f}{\partial x_j}(b_i)^2 = 1 \tag{24}$$

Hence, the FEP of tp_i can be expressed as

$$\begin{aligned} \mathrm{FEP}(tp_i) &= \mathop{\mathbf{E}}_{j}\left[\frac{\partial f}{\partial x_j}(a_i)^2 \cdot \frac{\partial f}{\partial x_j}(b_i)^2\right] \\ &= \frac{1}{n}(\frac{\partial f}{\partial x_1}(a_i)^2 \cdot \frac{\partial f}{\partial x_1}(b_i)^2 \cdots + \frac{\partial f}{\partial x_i}(a_i)^2 . \frac{\partial f}{\partial x_i}(b_i)^2 \\ &\cdots \frac{\partial f}{\partial x_n}(a_i)^2 \cdot \frac{\partial f}{\partial x_n}(b_i)^2). \end{aligned} \tag{25}$$

By following Equation (13), influence can be expanded as,

$$\begin{aligned} \mathrm{Inf}_i(f) &= \mathop{\mathbf{E}}_{x}\left[\frac{\partial f}{\partial x_i}(x)^2\right] = \sum_{S \ni i} \widehat{f}(\omega)^2. \\ &= \frac{1}{2^n}(\frac{\partial f}{\partial x_i}(0)^2 + \frac{\partial f}{\partial x_i}(1)^2 + \cdots + \frac{\partial f}{\partial x_i}(a_i)^2 + \\ &\frac{\partial f}{\partial x_i}(b_i)^2 + \cdots + \frac{\partial f}{\partial x_i}(2^{n-1})^2) \end{aligned} \tag{26}$$

In the influence expansion, it is known that $\frac{\partial f}{\partial x_i}(a_i)^2 = \frac{\partial f}{\partial x_i}(b_i)^2 = 1$. If all other terms are zero then the influence gets its minimum value as $1/2^{n-1}$. $\frac{\partial f}{\partial x_i}(x)^2 = 1$ means that $x_{j \neq i}$'s are masked in x such that output of the function follows x_i. The fact that Inf_i is high implies that $\Pr[f(x) = x_i]$ is high and it can be expected that $\frac{\partial f}{\partial x_j}(a_i)^2 \cdot \frac{\partial f}{\partial x_j}(b_i)^2 = 0$ for many $j \neq i$. In this case, $\mathrm{FEP}(tp_i)$ is expected to be smaller. □

Based on Proposition 1 and other concepts given so far, Definition 8 provides a new prioritization technique.

Definition 8
(Spectral Test Prioritization). Let $f(x) : \{-1,1\}^n \to \{-1,1\}$ be a Boolean expression and $\mathrm{Inf}_i(f)$ be the influence of input x_i on f. Let $TP = \{tp_1, tp_2, \ldots, tp_n\}$ be the MCDC test pairs for f and $T = \{t_1, t_2, \ldots, t_k\}$ be the test suite for f; PT be the set of permutations of T and ν the valuation function from PT to real numbers. The solution to the following problem gives the prioritized test suite:
Find $T' \in PT$ such that $\forall T'' \in PT, \nu(T') \geq \nu(T'')$ where

$$\nu(T) = \sum_{i,j \in [1,n]} [\mathrm{Inf}_i(f) - \mathrm{Inf}_j(f)](i - j).$$

This prioritization will be demonstrated using the aforementioned simple function $f(\mathbf{x}) = (x_1 + (x_2 \cdot x_3)) \cdot x_4$. Recall that MCDC pairs are computed as:

$$TP = \{(8,9), (12,14), (10,14), (1,9)\}.$$

After eliminating the repeating test cases, the unordered test suite would be written as follows:

$$T = \{8, 9, 12, 14, 10, 1\}.$$

To prioritize the test suite, the influence values for the inputs are calculated as shown in Table VI. Note that FEP values for the test pairs are also given in Table VI to demonstrate the negative

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

correlation. The following formula can be designated intuitively to compute the priorities:

$$p_i = \frac{1}{n} \cdot \frac{\mathrm{I}[f]}{\mathrm{Inf}_i(f)} \tag{27}$$

This formula fulfills the requirement of Definition 8 that p_i and Inf_i are inversely proportional and the priority for input i also represents the contribution of concerning influence to the total influence. Applying Formula (27), the test pairs can be assigned the priorities as shown below:

$$T = \{(8,9)^{0.833}, (12,14)^{2.5}, (10,14)^{2.5}, (1,9)^{0.625}\}$$

Note that priorities are related to the inputs and therefore they are directly assigned to the test pairs. It is assumed that test cases have the same priority values with their associating test pairs, i.e. $\mathrm{FEP}(tp_i) = \mathrm{FEP}(a_i) = \mathrm{FEP}(b_i)$ for $tp_i = (a_i, b_i)$. By eliminating the repeating test cases, the resulting test suite is obtained. If there exists more than one test cases having different priorities, the highest priority is selected. For example, $9^{0.625}$ is eliminated whereas $9^{0.833}$ is placed in the test suite:

$$T = \{12^{2.5}, 14^{2.5}, 10^{2.5}, 9^{0.833}, 8^{0.833}, 1^{0.625}\}$$

This test suite first applies test inputs 12 and 14 of the test pair 2 for testing condition x_2. Then by applying 10, pair 3 for condition x_3 is tested. Similarly, pair 1 and pair 4 are tested for the conditions x_1 and x_4 respectively.

Table VI. FEP and Influence values for $f(\mathbf{x}) = (x_1 + (x_2 \cdot x_3)) \cdot x_4$. $\mathrm{I}[f] = 1.25$ and Corr(FEP,Inf)=-0.99.

	x_1	x_2	x_3	x_4
Inf	0.375	0.125	0.125	0.625
p_i	0.833	2.5	2.5	0.5
FEP	0.621	0.862	0.862	0.783

Below, the effect of the proposed prioritization method will be demonstrated using only one fault hypothesis, ORF for the sake of simplicity. According to ORF, there exist three mutants of expression f as given below:

$$M_1(\mathbf{x}) = (x_1 \cdot (x_2 \cdot x_3)) \cdot x_4 \tag{28}$$

$$M_2(\mathbf{x}) = (x_1 + (x_2 + x_3)) \cdot x_4 \tag{29}$$

$$M_3(\mathbf{x}) = (x_1 + (x_2 \cdot x_3)) + x_4 \tag{30}$$

Table VII shows the values that f and its three mutants evaluate to against the test suite. Note that a test case kills the mutant if f and its mutant produce opposite values under a test case. If a mutant is killed by a test case, the corresponding value is shown with a star character in the table. For example, test case input 12 kills M_2 and M_3. Therefore, the FEP of test case 12 is $2/3$. On the other hand, the success of the prioritization technique can be measured how early the potential faults can be detected. As seen in Table VII, the first fault corresponding to mutant M_1 can be detected by the second test case, 14. However, the second and third faults (M_2 and M_3) can be detected by the first test case, 12. For the unprioritized test suite (test order is 8,9,12,14,10,1), the same analysis would result such that the faults can be detected by the 2$^{\text{nd}}$,3$^{\text{rd}}$ and 1$^{\text{st}}$ test cases. The original and prioritized testing processes are compared using a boxplot shown in Figure 5. The boxplot illustrates that prioritized test suite has a better detection time on average. More comprehensive evaluations with a relatively larger system will be demonstrated in the next section.

When an entire software or system is considered, there may exist several decisions represented by Boolean functions. For each decision, MCDC test cases are prioritized as inversely proportional to the influence values as explained before. If there are common test cases among several Boolean functions, the priority of any repeating test case needs to be modified such that the resulting priority will be the sum of priorities of all the repeating test cases. For example, if there are two test pairs

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

Table VII. Success of test cases for the three ORF mutants

$\mathbf{x}$	12	14	10	9	8	1
$f(\mathbf{x})$	F	T	F	T	F	F
$M_1(\mathbf{x})$	F	F*	F	F*	F	F
$M_2(\mathbf{x})$	T*	T	T*	T	F	F
$M_3(\mathbf{x})$	T*	T	T*	T	T*	T*
FEP	2/3	1/3	2/3	1/3	1/3	1/3

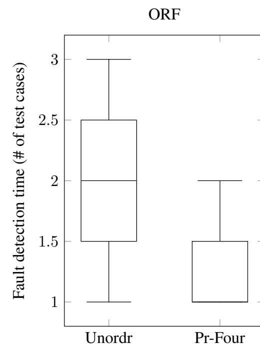

Figure 5. Comparison of unordered and ordered test suites in terms of fault detection times for ORF hypothesis.

that belong to two different decisions, *e.g.* $(2,5)^2$ and $(2,7)^3$, the prioritized corresponding portion of the test suite would be $T = \{2^5, 7^3, 5^2\}$. This summation allows to give higher priority to a test case that covers many entities, which is similar to the algorithm presented in [36].

The entire approach is summarized with Algorithm 1. Hereby, the input of algorithm consists of all decisions in a program or specifications expressed with Boolean functions. The output is the prioritized test suite.

6. EVALUATIONS

This section presents the evaluations of the proposed approach. First, Proposition 1 is checked by exploiting a brute-force approach. A large number of Boolean functions with 3 to 8 inputs are generated and the correlation between FEP and influence is validated against each function. The generic form of the Boolean functions is as follows:

$$y = ((((x_a \oplus z) \odot (x_b \oplus z)) \odot (x_c \oplus z)) \odot (x_d \oplus z)) \odot \cdots$$

where $a, b, c, d, \ldots \in [0, n-1]$, $z = \{0, 1\}$ and $\odot = \{\texttt{AND}, \texttt{OR}\}$. It is clear that the total number of generated functions is $n!\, 2^n\, 2^{n-1}$ and it can be computed as shown in Table VIII. All evaluations are performed on an Intel$^{\circledR}$ Core$^{\text{TM}}$ $i7-2600$ @ 3.40GHz CPU, 16 GB RAM, Linux version 2.6.32-35-server (gcc version 4.4.3) standard computer. A computer program runs the automated process such that Boolean functions are repeatedly generated and then their Fourier coefficients, including the influences are computed, MCDC test cases are derived and mutation analyses are performed to calculate the FEP values. The correlation between FEP and influence is also computed for each generated function. All the evaluations are performed under 2-fault assumption. This means that for a 3-input Boolean function, single fault or 2 simultaneous faults are possible. Faults are generated under ORF, CDF, CCF and ENF classes. Table VIII shows the results. As seen, average correlations always remain at -1.0.

To evaluate the entire approach, the set of predicates taken from Airborne Traffic Alert and Collision Avoidance System (TCAS-II) is used. They are shown in Table IX. TCAS-II specifications

Copyright © 2017 John Wiley & Sons, Ltd.

Algorithm 1 Prioritize_Test_Suite($allDecisions$)

```
Initialize all TP_{i∈{1,...,k}} ← ∅
Initialize all T_{i∈{1,...,k}} ← ∅
Initialize final test suite: T ← ∅
for all f in allDecisions do
    Find the set of MCDC test pairs TP for expression f
    Calculate influence values Inf(f)
    Assign priorities to the test pairs: p_i = (1/n) · I[f] / Inf_i(f)
end for
for all TP_{i∈{1,...,k}} do
    for all tp^p = (a^p, b^p) ∈ TP_i do
        if a ∉ T_i then T_i ← T_i ∪ {a^p}
        else
            Update the priority r of a^r in T_i : r ← max{r, p}
        end if
        if b ∉ T_i then T_i ← T_i ∪ {b^p}
        else
            Update the priority r of b^r in T_i : r ← max{r, p}
        end if
    end for
end for
for all T_{i∈{1,...,k}} do
    for all t^p ∈ T_i do
        if t ∉ T then T ← T ∪ {t^p}
        else
            Update the priority r of t^r in T_i : r ← r + p
        end if
    end for
end for
Order test suite T with respect to the priorities.
return T
```

Table VIII. Evaluations: Average correlations and computation times

# of inputs	# of Boolean functions generated	Average Correlation	Computation time
3	192	-1.000	0.001 minutes
4	3072	-1.000	0.170 minutes
5	61440	-1.000	0.517 minutes
6	1474560	-1.000	54 minutes
7	41287680	-1.000	1924 minutes
8	1321205760	-1.000	26 hours

vary from 5 to 14 variables. They were first introduced by Weyuker et.al [37] and have been commonly used as a benchmark for test generation techniques [15, 25, 26, 37, 38]. According to the table, there exist 14 conditions and 20 decisions. The number of conditions and the number of MCDC test cases generated for each decision are shown in Table X. From the table, it can be seen that there are totally 253 test cases generated for 20 specifications. After eliminating the repeating test cases, however, the size of test suite reduces to 134. For comparison, an existing MCDC prioritization algorithm is used [36]. This algorithm prioritizes the test cases based on their contribution values. The algorithm takes the unordered test suite and iteratively selects the strongest test case and adds it to the ordered test suite. The contribution of a test case t is the number of MCDC pairs completed by t plus the number of entries and exits covered by t and still uncovered

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

Table IX. TCAS-II Specifications.

1	$(ab)'(de'f' + d'ef' + d'e'f')(ac(d+e)h + a(d+e)h' + b(e+f))$
2	$(a((c+d+e)g + af + c(f+g+h+i))$ $+(a+b)(c+d+e)i)(ab)'(cd)'(ce)'(de)'(fg)'(fh)'(fi)'(gh)'(hi)'$
3	$(a(d' + e' + de(f'ghi' + g'hi)'(f'glk + g'i'k)')$ $+(f'ghi' + g'hi)'(f'glk + g'i'k)'(b + cm' + f))(ab'c' + a'bc' + a'b'c)$
4	$a(b' + c')d + e$
5	$a(b' + c' + bc(f'ghi' + g'hi)'(f'glk + g'i'k)') + f$
6	$(a'b + ab')(cd)'(fg'h' + f'gh' + f'g'h')(jk)'((ac + bd)e(f + (i(gj + hk))))$
7	$(a'b + ab')(cd)'(gh)'(jk)'((ac + bd)e(i' + g'k' + j'(h' + k')))$
8	$(a'b + ab')(cd)'(gh)'((ac + bd)e(fg + f'h))$
9	$(cd)'(e'fg'a'(bc + b'd))$
10	$ab'c'de'f(g + g'(h+i))(jk + j'l + m)'$
11	$ab'c'((f(g + g'(h+i)))' + f(g + g'(h+i))d'e')(jk + j'lm')'$
12	$abc'(f(g + g'(h+i)))(e'n' + d) + n'(jk + j'lm')$
13	$a + b + c + c'd'efg'h' + i(j+k)l'$
14	$ac(d+e)h + a(d+e)h' + b(e+f)$
15	$a((c+d+e)g + af + c(f+g+h+i))(a+b)(c+d+e)i$
16	$a(d' + e' + de(f'ghi' + g'hi)'(f'glk + g'i'k)') + (f'ghi' + g'hi)'(f'glk + g'i'k)'(b + cm' + f)$
17	$(ac + bd)e(f + (i(gj + hk)))$
18	$(ac + bd)e(i + g'k' + j'(h' + k'))$
19	$(ac + bd)e(fg + f'h)$
20	$e'fg'a'(bc + b'd)$

Table X. Number of inputs and size of MCDC test suites for TCAS-II expressions.

TCAS#	1	2	3	4	5	6	7	8	9	10
n	7	9	12	5	9	11	10	8	7	13
# of tests	8	12	16	8	12	15	15	11	9	16
TCAS#	11	12	13	14	15	16	17	18	19	20
n	9	14	12	7	9	12	11	10	8	7
# of tests	11	17	15	10	12	17	15	14	11	9

by the test cases in the ordered test suite. This algorithm is called as *Pr-Covr* in the remaining of the text. Figure 6 shows the evaluation results. The figure consists of five box plot diagrams. The four diagrams on left hand side demonstrate the fault detection times of the test suites against the faults systematically generated by 4 fault hypotheses ORF, CCF, CDF and ENF. The right hand side diagram illustrates the case of these fault hypotheses all together. In each diagram, there exist three box plots related to the unordered initial test suite, ordered test suite by *Pr-Covr* and the ordered suite by the proposed algorithm *Pr-Four*, respectively. Fault detection times are such that when test cases are sequentially applied to a faulty software, the index of first test case that reveals the fault is considered as the detection time. If the time duration of applying one test is known, this value can be straightforwardly transformed to the real time as well. Fault detection times are computed against almost 4000 faulty versions of the entire specifications and box plots explicitly show the first, second (median) and third quartiles. For example, the median values for the right hand side diagram are 65, 43 and 24 for the unordered, *Pr-Covr* and *Pr-Four* test suites respectively. The plots show that *Pr-Four* has the fastest fault detection capability for all fault classes. This means that in case of unordered test suite and *Pr-Covr* any fault can be detected at 65th and 43rd test cases, whereas in *Pr-Four* a fault can be detected at 24th test case on average.

Another common metric to measure the effectiveness of a test suite is the weighted Average of the Percentage of Faults Detected (APFD). APFD measures how quickly a test suite detects faults. For a test suite T with n test cases and m number of faults, APFD metric is defined as follows [5]:

$$\text{APFD} = 1 - \frac{TF_1 + TF_2 + \ldots + TF_m}{n\,m} + \frac{1}{2n} \tag{31}$$

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

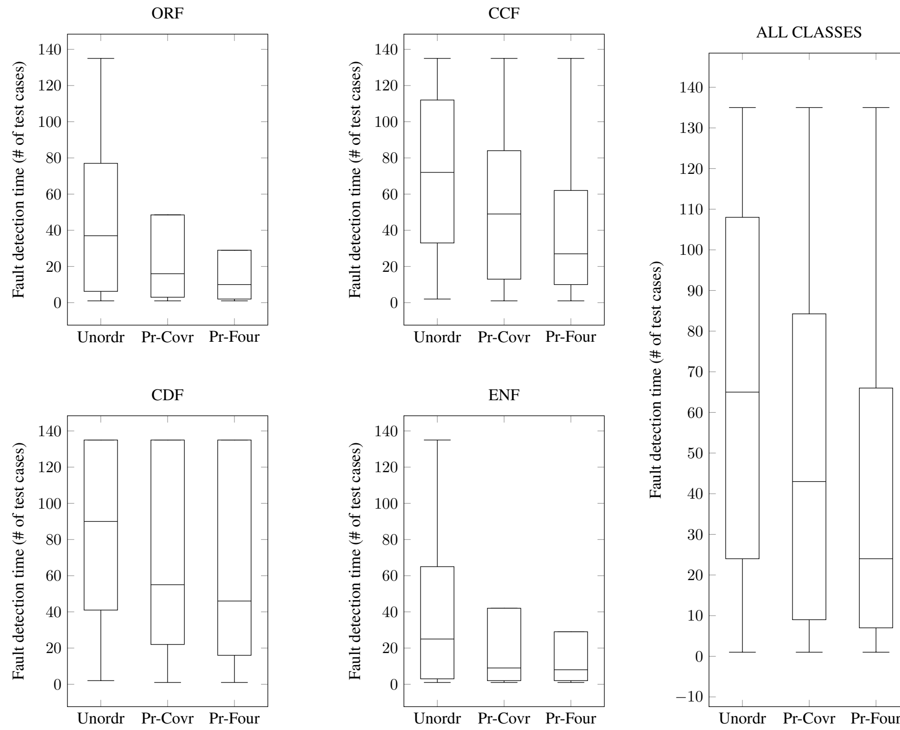

Figure 6. Comparison of three methods in terms of fault detection times

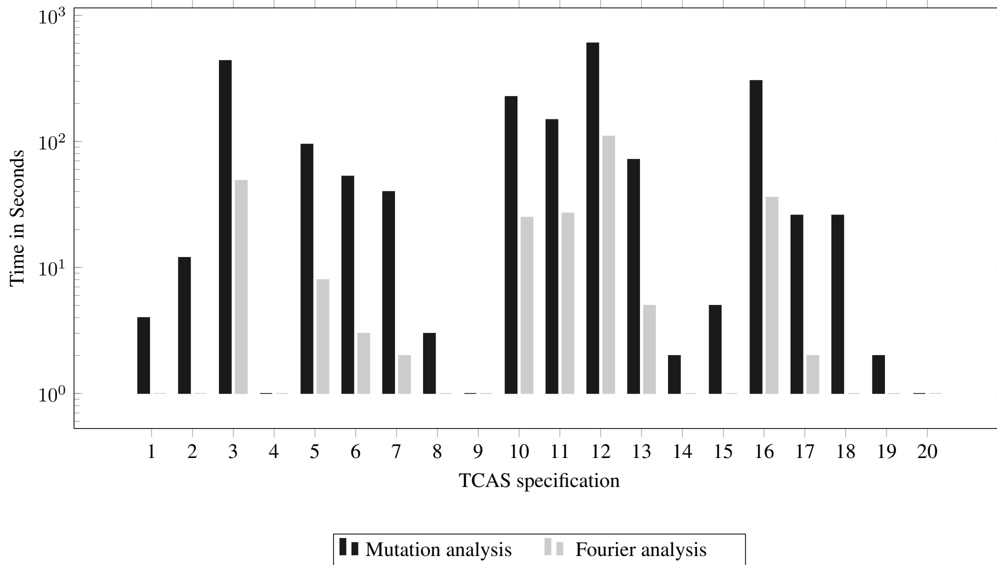

Figure 7. Comparison of time costs for mutation and Fourier analyses

where TF_i is the position of the first test case that reveals fault i. The APFD values for the three test suites, unordered, prioritized with *Pr-Covr* and prioritized by *Pr-Four* are given in Table XI. As expectedly from Figure 6, *Pr-Four* outperforms the others.

Compared with *Unordr* and *Pr-Covr*, *Pr-Four* is a highly time consuming algorithm. For FEP prioritization, mutation analysis is the bottleneck. For *Pr-Four*, however, the bottleneck is Fourier analysis. Therefore, Figure 7 compares the time costs of mutation and Fourier analyses of TCAS-II expressions. Mutation analysis is performed under 2-fault assumption and again using the same four

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

Table XI. APFD values of the three test suites under test (4000 faulty versions are generated under 4 fault classes).

Prioritization	Unordr	Pr-Covr	Pr-Four
APFD	0.514	0.618	0.684

classes, ORF, CCF, CDF and ENF. Note that the figure uses logarithmic y-axis and Fourier analysis outperforms the mutation analysis, particularly in relatively large expressions. For even larger expressions, it is also possible to approximate the Fourier coefficients as remarked in Section 4.1, which makes the approach more efficient.

Both the exact and approximate computations of influence values depend on a simple Boolean arithmetic and they are easy to implement as opposed to the mutation analysis. Therefore, the proposed prioritization scheme could be incorporated into any existing utilities.

7. THREATS TO VALIDITY

Four types of threats can be considered [39]: Conclusion, construct, internal and external validities for the validity of the proposed prioritization approach.

Conclusion validity is mitigated by experimenting a possibly large number of auto-generated Boolean functions and calculating the correlations between FEP and Inf values. However, up to 8-input functions are handled due to the computation overhead. Regarding the construct validity, the results are compared with respect to well-established metrics in the literature. It is also assumed that all test cases are equal in cost. This is generally true in that each test case takes an almost fixed amount of time to execute. Another assumption is that faults are supposed to have the same severity. It is likely that some faults can be more severe, and therefore the test cases that reveal such faults should have higher priorities.

To overcome internal threats, a previously published system is utilized in the experiments and the same tool is used to perform all the calculations like Fourier analysis, generation of MCDC test cases, mutation analysis and performance comparisons.

External validity is related to the generalization of the approach. The experiments are performed on relatively large yet a single system TCAS-II, which can be considered as a serious threat. However, the success of FEP prioritization shown in the literature and the negative correlation validated by a large number of auto-generated functions help to mitigate this threat.

The limitation of the proposed approach is the time consumption of Fourier analysis. The time complexity of the traditional Fourier computation is given as $\mathcal{O}(n2^n)$ and the influence computation then would be $\mathcal{O}(n^2 2^n)$. However, as demonstrated with Figure 7, Fourier has still better performance than the mutation analysis. Moreover, approximate calculation of Fourier coefficients mitigates this limitation.

8. RELATED WORK

Software testing is an expensive process and it can consume half of the whole development process and sometimes more than that in case of the need for higher levels of reliability [1, 40]. Today many software systems require large amount of tests that cannot be completed due to strict time and cost limitations. It is therefore of paramount importance to reduce the test suite, or schedule the tests such that the highest order tests are run first to increase the likelihood of detecting faults earlier. To this end, test case selection, test suite reduction and test case prioritization methods are suggested. Among them, the aim of test case prioritization is to rank the test cases to reveal faults earlier in order to reduce the cost of testing. There are numerous test case prioritization methods proposed in the literature [2, 3, 4, 5, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55].

Copyright © 2017 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

DOI: 10.1002/stvr

Rothermel was the first to formally define the test case prioritization problem as given in Definition 7 with the focus on reducing the cost per fault coverage [2]. With the growth of the size of software and consequently amount of testing efforts, test case prioritization has become more important, which has taken the attention of many researchers. Almost all studies address the issue of regression testing that is the re-execution of the tests that have already been conducted while software evolves. However, as the amount of efforts and cost in time and budget to run a complete test suite can be tremendous in recent times, it is undisputedly beneficial even for the initial testing. Elbaum presents six heuristics for prioritization based on the coverage performance fed back from the prior executions of the tests [4]. These heuristics mainly rely on the bases of *Statement Coverage*, *Branch Coverage* and *Fault Exposing Potential (FEP)*. Prioritizing by the FEP of the test cases however differs from the other two techniques such that FEP is a probability that the test case reveals the potential faults. This value must be estimated under certain fault classes or assumptions. To estimate the FEP values, there exists a well-studied mutation analysis method. Several works utilize mutation analysis to estimate the FEP and conclude that FEP prioritization is an effective technique [2, 3, 4, 5, 55].

Mutation testing is based on two main assumptions about the types of errors which typically occur in software. The first is the *"Competent Programmer Hypothesis"* which essentially states that programmers write programs that are reasonably close to the desired program. Second, the *"Coupling Effect"* postulates that all complex faults are the product of one or more simple fault(s) occurring within the software. Both of these assumptions are essential in demonstrating that, by making simple substitutions in the software under test, mutation replicates the types of errors typically made by developers. A recent work [56] empirically evaluates the mutation testing particularly taking into account safety-critical software. As noted in the aforementioned works, in spite of its efficacy, the FEP prioritization technique seems to be more complicated than other techniques due to the expense of the mutation analysis. One contribution of the proposed approach is to facilitate FEP prioritization by moving the problem to a different domain.

Boolean expressions are frequently of interest, for example in testing the requirements or the structure of the software. In this case, one can benefit from many useful and powerful techniques like Boolean derivative calculus and Fourier analysis of Boolean functions. Boolean derivative calculus is particularly used in hardware testing [6, 7, 8, 9] and it is rarely seen in software testing except for the MCDC test suite definition [10, 12]. However, in these works the boolean derivation is only used to give a formal definition and any further mathematical analysis is not presented.

Spectral analysis of Boolean functions is also another powerful technique. In the literature, Fourier analysis, Walsh or Walsh-Hadamard transformations, Reed-Muller transformation are all used to specify the spectral analysis and they have been well-known for more than thirty years. Although the technique has a wide application area in mathematics, physics and engineering, its application in computer science seems relatively limited. Some fields that the Fourier helped so far are the analysis of error-correcting codes, cryptography, graph theory and quantum computing. Spectral analysis of Boolean functions has attracted a great attention from computer scientists in the last decade [29, 30, 31]. This is due to some nice theorems such as Kahn-Kalai, Arrow's and Peres's theorems, and also its contribution in the development of social choice theory. The influence of Boolean variables and noise sensitivity have also been studied by several papers [29, 31, 57].

There are few studies on the application of Fourier in testing. An early work of Susskind proposes testing the circuits through the Fourier coefficients [58]. A more recent and comprehensive work of Falkowski presents the theory of Walsh transformation and its applicability on testing digital circuits [59]. Yogi proposes an Automatic Test Pattern Generation (ATPG) method for circuits [60, 61]. Here, randomly generated test vectors under the stuck-at fault assumption are spectrally analyzed and the major components are extracted so that the vectors with a better fault coverage can be selected.

This work presents an idea and sound method for a better prioritization of MCDC test cases. Test suite prioritization for MCDC coverage was previously studied by Jones and Harrold [36]. They present a test suite prioritization algorithm assigning contribution values to the test cases depending on their coverage. The contributions are computed on the basis of the coverage of the

Copyright © 2017 John Wiley & Sons, Ltd.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

entry and exit points and the statements in a program. Our proposed method, however provides a better prioritization of MCDC test cases taking into account also the FEP of test cases. Moreover, this is achieved by a relatively simpler Fourier analysis instead of a mutation analysis. As explained in the previous sections, the aim is to increase the likelihood of detecting the faults earlier so that the testing can be immediately paused to localize and correct the errors.

The benefit of the method is multi-fold: i) it provides an easy technique to order the test cases, ii) it is based on mathematics, iii) it outperforms the identical FEP prioritization and iv) it gives a useful insight how to incorporate Fourier analysis in software testing. A strong relationship between the influences of conditions on the decision and the fault exposing potentials of MCDC test cases under certain fault assumptions allows to reduce the prioritization problem to the relatively simple computation of influence variables.

9. CONCLUSION

This work proposes a Fourier analysis-based prioritization method for MCDC test suites. The correlation between fault exposing potentials of the test cases and the influence values of their associating input conditions under certain fault hypotheses allows to order test cases with respect to the influences only. This eliminates the expensive mutation analysis. The method relies on mathematics and its efficacy is demonstrated with the experiments on a large number of systematically generated Boolean functions and also a benchmark example from industry. The method provides a useful insight into how the spectral analysis benefits test case prioritization. It can be further investigated to cover other fields of testing as well. As the Fourier analysis of Boolean functions reveals many useful features of them, it seems promising for all fault based Boolean testing methods not only to prioritize tests but also to help with more efficient test case generation. In the most general sense, the term “more efficient” must be understood as faster generation of a minimal test suite that has a better fault exposing potential. To this end, a more in depth study to uncover the potential relations between the Fourier coefficients together with the noise-related parameters and the various fault hypotheses could be of considerable interest in the future research.

REFERENCES

1. Tassey G. The economic impacts of inadequate infrastructure for software testing. *Technical Report* 2002, doi:10.1080/10438590500197315. URL http://www.nist.gov/director/prog-ofc/report02-3.pdf.
2. Rothermel G, Untch RH, Chu C, Harrold MJ, Society IC. Prioritizing Test Cases For Regression Testing Prioritizing Test Cases For Regression Testing. *Software Engineering, IEEE Transactions on* 2001; **27**(10):929–948.
3. Rothermel G, Untch RH, Chu C, Harrold MJ. Test Case Prioritization: An Empirical Study. *Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference on*, 1999; 179 – 188.
4. Elbaum S, Malishevsky A, Rothermel G. Incorporating varying test costs and fault severities into test case prioritization. *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, 2001; 329–338, doi:10.1109/ICSE.2001.919106. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=919106.
5. Elbaum S, Rothermel G, Malishevsky AG, Member S. Test Case Prioritization : A Family of Empirical Studies Test Case Prioritization : A Family of Empirical Studies. *Software Engineering, IEEE Transactions on* 2002; **28**(2):159–182, doi:10.1109/32.988497.
6. George W, Smith J. Fault Detection Test Derivation Using Boolean Difference Techniques. *ACM '72 Proceedings of the ACM annual conference*, 1972; 369–378.
7. Reed IS. Boolean Difference Calculus and Fault Finding 1973, doi:10.1137/0124014.
8. Changqian W, Chenghua W. A Method for Logic Circuit Test Generation Based on Boolean Partial Derivative and BDD. *2009 WRI World Congress on Computer Science and Information Engineering*, Ieee, 2009; 499–504, doi:10.1109/CSIE.2009.44. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5170892.
9. Palaka S, Rao KS. Fault Tolerant Logic Design For Multiple Faults In Combinational Circuits. *International Journal of Engineering Research and Applications (IJERA)* 2012; **2**(4):1237–1242.
10. Chilenski JJ, Miller SP. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal* 1994; **9**(5):193, doi:10.1049/sej.1994.0025.
11. RTCA. DO-178B: Software considerations in airborne systems and equipment certification. *Technical Report*, RTCA Inc. December 1992.
12. Chilenski JJ. An investigation of three forms of the modified condition decision coverage (MCDC) criterion. *Security* 2001; (April). URL http://www.stormingmedia.us/40/4002/A400293.pdf.

Copyright © 2017 John Wiley & Sons, Ltd.
Prepared using stvrauth.cls

13. Ammann P, Offutt J. *Introduction to Software Testing*. 1 edn., Cambridge University Press: New York, NY, USA, 2008.
14. Hayhurst KJ, Veerhusen DS, Chilenski JJ, Rierson LK. *A Practical Tutorial on Modified Condition/Decision Coverage*. May, 2001.
15. Badhera U. Fault Based Techniques For Testing Boolean Expressions: A Survey. *International Journal of Computer Science & Engineering Survey* 2012; **3**(1):81–90, doi:10.5121/ijcses.2012.3108.
16. Kapoor K, Bowen JP. A formal analysis of mcdc and rcdc test criteria. *Softw. Test., Verif. Reliab.* 2005; **15**(1):21–40. URL http://dblp.uni-trier.de/db/journals/stvr/stvr15.html#KapoorB05.
17. Myers GJ, Myers GJ. *Art of Software Testing*. 1979.
18. Myers G. *The Art of Software Testing, Second edition*, vol. 15. 2004, doi:10.1002/stvr.322. URL http://www.noqualityinside.com/nqi/nqifiles/TheArtofSoftwareTesting-SecondEdition.pdf.
19. Ayav T, Belli F. Boolean differentiation for formalizing Myers' cause-effect graph testing technique. *Proceedings of the 15th IEEE International Conference on Software Quality, Reliability and Security*, Vancouver, BC, 2015; 138–143, doi:10.1109/QRS-C.2015.31.
20. (Cast) CAST. Rationale for Accepting Masking MC/DC in Certification Projects 2001; .
21. Paul TK, Lau MF. A systematic literature review on modified condition and decision coverage. *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14*, 2014; 1301–1308, doi:10.1145/2554850.2555004. URL http://dl.acm.org/citation.cfm?doid=2554850.2555004.
22. Demillo Ra, Lipton RJ, Sayward FG. Hints on Test Data Selection. *Computer* 1978; **11**(4):34–41.
23. Kuhn R. Fault classes and error detection capability of specification-based testing. *ACM Transactions on Software Engineering and Methodology* 1999; **8**(4):411–424, doi:10.1145/504087.504089.
24. Tsuchiya T, Kikuno T. On fault classes and error detection capability of specification-based testing. *ACM Transactions on Software Engineering and Methodology* 2002; **11**(1):58–62, doi:10.1145/504087.504089.
25. Lau MF, Yu YT. An extended fault class hierarchy for specification-based testing. *ACM Transactions on Software Engineering and Methodology* 2005; **14**(3):247–276, doi:10.1145/1072997.1072998.
26. Chen TY, Lau MF, Sim KY, Sun Ca. On detecting faults for boolean expressions. *Software Quality Journal* 2009; **17**(3):245–261, doi:10.1007/s11219-008-9064-5.
27. Kapoor K, Bowen JP. Test conditions for fault classes in Boolean specifications. *ACM Trans. Softw. Eng. Methodol.* 2007; **16**(3):10, doi:10.1145/1243987.1243988. URL http://portal.acm.org/citation.cfm?id=1243987.1243988{&}coll=portal{&}dl=ACM{&}idx=J790{&}part=transaction{&}WantType=Transactions{&}title=ACMTransactionsonSoftwareEngineeringandMethodology(TOSEM){&}CFID=16581986{&}CFTOKEN=87385715$\delimiter"026E30F$nhttp://portal.acm.org/ft{_}gateway.cfm?id=1243988{&}type=pdf{&}coll=portal{&}dl=ACM{&}CFID=16581986{&}CFTOKEN=87385715.
28. Chen Z, Chen TY, Xu B. A Revisit of Fault Class Hierarchies in General Boolean Specifications. *ACM Trans. Softw. Eng. Methodol.* 2011; **20**(3):13:1—-13:11, doi:10.1145/2000791.2000797. URL http://doi.acm.org/10.1145/2000791.2000797.
29. O'Donnell R. Some topics in analysis of boolean functions. *Proceedings of the fourtieth annual ACM symposium on Theory of computing - STOC 08* 2008; :569doi:10.1145/1374376.1374458. URL http://dl.acm.org/citation.cfm?doid=1374376.1374458.
30. O'Donnell R, Austrin P. Analysis of Boolean Functions: Notes from a series of lectures by Ryan O'Donnell. *Workshop on Computational Complexity* 2012; .
31. Wolf RD. A Brief Introduction to Fourier Analysis on the Boolean Cube. *Theory of Computing, Graduate Surveys* 2008; **1**(015848):1–20, doi:10.4086/toc.gs.2008.001. URL http://toc.cse.iitk.ac.in/articles/gs001/gs001.pdf.
32. Porwik P. Efficient sprectral method of identification of linear boolean function. *Control and Cybernatics* 2004; **33**(4):83–105.
33. Mitzenmacher M, Upfal E. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press: New York, NY, USA, 2005.
34. Weyuker E, Singh A. from a Boolean Specification. *Software Engineering, IEEE Transactions on* 1994; **20**(5):353 – 363.
35. Singh RK, Chandra P, Singh Y. An evaluation of Boolean expression testing techniques. *ACM SIGSOFT Software Engineering Notes* 2006; **31**(5):1, doi:10.1145/1163514.1163534.
36. Jones Ja, Harrold MJ. Test-suite reduction and prioritization for modified condition/decision coverage. *IEEE International Conference on Software Maintenance, ICSM* 2001; :92–103doi:10.1109/ICSM.2001.972715.
37. Weyuker E, Goradia T, Singh A. Automatically Generating Test Data from a Boolean Specification 1994.
38. Gargantini A, Fraser G. Generating minimal fault detecting test suites for general Boolean specifications. *Information and Software Technology* 2011; **53**(11):1263–1273, doi:10.1016/j.infsof.2011.06.008.
39. Wohlin C, Runeson P, Host M, Ohlsson M, Regnell B, Wesslen A. *Experimentation in Software Engineering*. Springer-Verlag, Berlin, Heidelberg, 2012.
40. Beizer B. *Software Testing Techniques*. Van Nost. Reinhold, U.S., 1990.
41. Srivastava PR. Test case prioritization. *Journal of Theoretical and Applied Information Technology* 2008; (December):178–181, doi:10.1109/ISISE.2008.106. URL http://www.jatit.org/volumes/research-papers/Vol4No3/Regression_Testing_Average_Percentage_of_Faults_Detected_APFD_Test_Cases.pdf.
42. Srivastava PR, Patel P, Chatrola S. Cause effect graph to decision table generation. *ACM SIGSOFT Software Engineering Notes* 2009; **34**(2):1, doi:10.1145/1507195.1507216.
43. Li Y, Yin Bb, Lv J, Cai Ky. Approach for Test Profile Optimization in Dynamic Random Testing. *IEEE 39th COMPSAC*, 2015; 466–471, doi:10.1109/COMPSAC.2015.257.

Copyright © 2017 John Wiley & Sons, Ltd.

DOI: 10.1002/stvr

44. Kavitha R. Test Case Prioritization for Regression Testing based on Severity of Fault. *International Journal on Computer Science and Engineering* 2010; **02**(05):1462–1466.
45. Gokce N, Belli F, Eminli M, Dincer BT. Model-based test case prioritization using cluster analysis: a soft-computing approach. *Turkish Journal of Electrical Engineering & Computer Sciences* 2015; **23**:623–640, doi:10.3906/elk-1209-109. URL http://online.journals.tubitak.gov.tr/openDoiPdf.htm?mKodu=elk-1209-109.
46. Muthusamy T. A New Effective Test Case Prioritization for Regression Testing based on Prioritization Algorithm. *International Journal of Applied Information Systems (IJAIS)* 2014; **6**(7):21–26.
47. Srikanth H, Williams L, Osborne J. System test case prioritization of new and regression test cases. *2005 International Symposium on Empirical Software Engineering, ISESE 2005*, 2005; 64–73, doi:10.1109/ISESE.2005.1541815.
48. Hao DAN, Zhang L, Zhang LU, Rothermel G, Mei H. A Unified Test Case Prioritization Approach. *ACM Transactions on Software Engineering and Methodology* 2014; **24**(2).
49. Zhang L, Hou SS, Guo C, Xie T, Mei H. Time-aware test-case prioritization using integer linear programming. *Proceedings of the eighteenth international symposium on Software testing and analysis - ISSTA '09* 2009; :401–419doi:10.1145/1572272.1572297. URL http://portal.acm.org/citation.cfm?doid=1572272.1572297.
50. Lingming Zhang DH, Zhang L, Rothermel G, Mei H. Bridging the Gap between the Total and Additional Test-Case Prioritization Strategies. *ICSE '13 Proceedings of the 2013 International Conference on Software Engineering*, 2013; 192–201.
51. Mei L, Zhang Z, Chan WK, Tse TH. Test case prioritization for regression testing of service-oriented business applications. *Proceedings of the 18th international conference on World wide web - WWW '09*, 2009; 901–910, doi:10.1145/1526709.1526830. URL http://portal.acm.org/citation.cfm?doid=1526709.1526830.
52. You D, Chen Z, Xu B, Luo B, Zhang C. An empirical study on the effectiveness of time-aware test case prioritization techniques. *Proceedings of the 2011 ACM Symposium on Applied Computing*, 2011; 1451–1456, doi:10.1145/1982185.1982497. URL http://portal.acm.org/citation.cfm?id=1982497.
53. Catal C. On the application of genetic algorithms for test case prioritization: a systematic literature review. *Proceedings of the 2nd international workshop on evidential assessment of software technologies*, 2012; 9–14, doi:10.1145/2372233.2372238. URL http://dl.acm.org/citation.cfm?id=2372238.
54. Suri B, Singhal S. Analyzing test case selection & prioritization using ACO. *ACM SIGSOFT Software Engineering Notes* 2011; **36**(6):1, doi:10.1145/2047414.2047431.
55. Wang S, Buchmann D, Ali S, Liaaen M. Multi-Objective Test Prioritization in Software Product Line Testing : An Industrial Case Study. *SPLC '14 Proceedings of the 18th International Software Product Line Conference*, 2014; 32–41.
56. Baker R, Habli I. An empirical evaluation of mutation testing for improving the test quality of safety-critical software. *IEEE Transactions on Software Engineering* 2013; **39**(6):787–805, doi:10.1109/TSE.2012.56.
57. Kahn J, Kalai G, Linial N. The influence of variables on Boolean functions. *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science* 1988; doi:10.1109/SFCS.1988.21923.
58. Susskind AK. Testing by Verifying Walsh Coefficients. *IEEE Transactions on Computers* 1983; **32**(2):198–201.
59. Falkowski BJ. Spectral Testing of Digital Circuits. *VLSI Design* 2002; **14**(1):83–105, doi:10.1080/10655140290009828. URL http://www.hindawi.com/journals/vlsi/2002/587524/abs/.
60. Yogi N. Spectral Methods for Testing of Digital Circuits. PhD Thesis, Auburn University 2009.
61. Yogi N, Agrawal V. High-Level Test Generation for Gate-Level Fault Coverage. *Proc. 15th IEEE North Atlantic Test Workshop* 2006; URL http://www.eng.auburn.edu/~vagrawal/TALKS/NATW06_Yogi.pdf.

Copyright © 2017 John Wiley & Sons, Ltd.