Generating Minimal Test Set Satisfying MC/DC Criterion via SAT Based Approach

Ling Yang[1,3], Jun Yan[1,2,3,†] , and Jian Zhang[1,3,†]
1. State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences, Beijing, China
2. Technology Center of Software Engineering
Institute of Software, Chinese Academy of Sciences, Beijing, China
3. University of Chinese Academy of Sciences, Beijing, China
Email: {yangling, yanjun, zj}@ios.ac.cn

ABSTRACT

The Modified Condition/Decision Coverage (MC/DC) is a test criterion proposed by NASA to measure the test set adequacy, especially in safety-critical systems. Existing works on test case generation for this criterion usually make use of greedy or meta-heuristic search techniques to construct a small test set. This paper studies the problem of generating optimal test set satisfying this criterion. It enumerates the size of test set from the theoretic lower-bound $m+1$ to upper-bound $2m$ where m is the number of conditions that can be MC/DC covered, and then employs a SAT solver to decide whether there exists a test set with a given size, until a test set is found. We apply this method to benchmarks with no more than 25 conditions including dozens of instances collected from related works and millions of randomly generated ones. The experimental results indicate that most (more than 99%) of these instances can be MC/DC covered by a test set with only $m+1$ test cases. Therefore, our SAT-based approach is efficient for generating optimal MC/DC test sets, since for most of the instances, it only invokes the SAT solver just once in the process of searching for minimal test set.

CCS CONCEPTS

• **Software engineering → Testing and Debugging**;

KEYWORDS

MC/DC, test data generation, SAT-based, Boolean expressions

ACM Reference Format:
Ling Yang, Jun Yan, and Jian Zhang. 2018. Generating Minimal Test Set Satisfying MC/DC Criterion via SAT Based Approach. In *SAC 2018: SAC 2018: Symposium on Applied Computing , April 9–13, 2018, Pau, France.* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3167132.3167335

[†]Corresponding authors.

SAC 2018, April 9–13, 2018, Pau, France

ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/10.1145/3167132.3167335

1 INTRODUCTION

Boolean expressions are widely used in software systems. How to ensure the correctness of Boolean expressions, and how to generate test cases from Boolean specifications, are challenging problems in testing, especially for safety-critical systems. Consider a Boolean expression E that contains n variables, in theory, the number of value combinations of these n variables will be 2^n. It's a huge number and makes the multiple condition testing expensive while n is quite big. A critical problem for such multi-condition testing is reducing the number of test cases by test data selection, and meanwhile guaranteeing the quality of fault detection. Many test coverage criteria have been designed as a practical way of measuring the effectiveness of testing. Modified condition/decision coverage (MC/DC), which was proposed by NASA in 1994 [5], is one of the most well-known adequacy criteria to ensure all conditions and decisions be tested according to some specific rules. MC/DC is a white-box testing criterion and its idea is that a decision must be tested by demonstrating that each condition can influence the outcome of the decision independently.

The requirements of MC/DC criterion is complex that need to use a pair of test cases to cover each condition individually, and it is not easy to construct a proper test set manually for system testers. How to automatically generate the test set satisfying the criterion is a key problem of MC/DC testing. In the recent decades, several approaches [1, 4, 11] based on greedy or meta-heuristic search strategy have been proposed for test data generation of MC/DC criterion. However, the algorithms in these works are incomplete, that is, they can not guarantee that the solutions are optimal. Hence, till now there are no works addressing the optimal test set of MC/DC. This paper tries to make use of complete search techniques to find the minimal test set satisfying the MC/DC criterion.

Since we consider MC/DC testing of Boolean expressions in this paper, a reasonable way is to make use of achievements in SAT problem (Propositional Satisfiability Problem), where many mature complete solvers have been developed. As we know, finding the minimal test set is an optimization problem. In our approach we transform the decision problem of whether there exists a solution with some given size into SAT problem, and invoke SAT solvers to find the minimal size. We implement the proposed method into an automatic tool, which can efficiently find the optimal solutions for MC/DC criterion in small-scale (less than 20 Boolean variables)

within acceptable time. Our experimental results also first reveal a phenomenon that most of the decisions can be MC/DC covered by $m + 1$ test cases that is the lower bound in theory, where m is the number of conditions.

The rest of this paper is organized as follows. Section 2 gives some basic definitions and the notation mainly about MC/DC criterion. Section 3 describes our approach to generate the optimal test set of MC/DC. Section 4 shows the experimental setup and gives the three kinds of benchmarks that are used in the experiment. Section 5 shows the experimental results by our tool based on our approach. Section 6 presents some discussions about the definition of MC/DC and the minimum size of test set of MC/DC. Section 7 discusses previous work related to the generation of test set for MC/DC. Section 8 summarizes and concludes the paper.

2 BACKGROUND

In this section, we provide several basic definitions which are used throughout this paper, mainly about MC/DC.

2.1 Notation and terminology

Since predicates in program source code, as well as logic expressions in software specifications, can be formalized as Boolean expressions via using different transition rules or techniques, we only consider Boolean expressions as input data in this paper. The definitions of Boolean expressions are given by definition 2.1.

Definition 2.1 (Boolean expression). *A **Boolean expression** is an expression that evaluates to a value of either True or False. Generally, T or 1 is used for True and F or 0 for False. We denote the Boolean operators* `NOT`, `AND`, `OR`, `XOR` *(exclusive-or),* `IMPLY` *(implication) and* `EQU` *(equivalence) by* $\neg$, $\wedge$, $\vee$, $\oplus$, $\rightarrow$, *and* $\leftrightarrow$, *respectively.*

The descriptions for condition and decision have been provided by [7, 8] for better understanding the concept of MC/DC.

Definition 2.2 (Condition and Decision). *A **condition** is a Boolean variable containing no Boolean operators. A **decision** is a Boolean expression composed of conditions and zero or more Boolean operators.*

Note that our definition is slightly different from that of the document [8], in which a Boolean variable appearing more than once in a decision is regarded as distinct conditions, while in this paper we believe that a Boolean variable can only take one value in a test, therefore we treat the multiple occurrences of a variable as one condition. Take the decision $S = (a \wedge b \wedge c) \vee (\neg c \wedge d \wedge e)$ as an example, according to our definition it has five conditions; while according to the definition of the document [8] it has six conditions for the variable c occurs twice. More about the difference of definitions of condition, please refer to section 6.1.

Definition 2.3 (Test set for decision). *Let $S(x_1, x_2, \ldots, x_n)$ denote a decision, where the Boolean-domain variable x_r $(1 \leq r \leq n)$ indicates the r'th condition. A **test case**, also called a **test point**, for the decision S is a truth vector $tc = (v_1, v_2, \ldots, v_n)$, where $v_r \in \{0, 1\}$ $(1 \leq r \leq n)$ is the value assigned to the condition x_r. A **test set** with k test cases $TS = \{tc_1, tc_2, \ldots, tc_k\}$ is a collection of test cases for a decision and k is the **size** of the test set. We use $S(tc_i)$ to represent the value of decision S under test case tc_i, or the outcome of decision S.*

2.2 The MC/DC criterion

As its name implies, the MC/DC (Modified Condition/Decision Coverage) criterion consists of two related criteria: the condition coverage (CC) and the decision coverage (DC) criterion.

CC Every condition in a decision has taken on all possible outcomes at least once.

DC Every decision has taken on all possible outcomes at least once.

However, DC only considers the outcomes of the decision that have taken both *True* and *False* values, while CC does not take the outcomes of the decision into consideration. Hence, MC/DC criteria was proposed and defined as follows [8]:

Definition 2.4 (MC/DC). *Every condition in a decision has taken on all possible outcomes at least once, every decision has taken on all possible outcomes at least once, and each condition in a decision has been shown to **independently affect** that decision's outcome. A condition is shown to independently affect a decision's outcome by varying just that condition while holding fixed all other possible conditions.*

Here, we give the formal description of this criterion. Consider the decision $S(x_1, x_2, ..., x_n)$ with n conditions. Let $tc_1 = (v_{1,1}, v_{1,2}, \ldots, v_{1,n})$ and $tc_2 = (v_{2,1}, v_{2,2}, \ldots, v_{2,n})$ indicate two test cases for decision S. The two test cases tc_1 and tc_2 ***MC/DC cover*** the r'th condition x_r of S if

(1) the two test cases tc_1 and tc_2 are different only in the r'th condition, i.e., $v_{1,r} \neq v_{2,r}$ and for all $\ell \neq r$ $(1 \leq \ell \leq n)$, $v_{1,\ell} = v_{2,\ell}$;
(2) the decision takes different outcomes under these two test cases, i.e., $S(tc_1) \neq S(tc_2)$.

We call these two test cases $\{tc_1, tc_2\}$ as the ***MC/DC pair*** for condition x_r. Since some of conditions may be bypassed (e.g., the condition x of decision $0 \wedge x$) and cannot affect the outcomes of the decision, not all the conditions can be MC/DC covered.

Example 1. Consider the decision $S = (a \wedge b \wedge c) \vee (\neg c \wedge d \wedge e)$ with 5 conditions.

A test set shown in Table 1 satisfies MC/DC criterion. All of the conditions a, b, c, d and e have taken all possible outcomes (0 and 1) at least once, and the outcomes of the decision have taken the value 0 and 1. For condition a, the pair of test cases tc_1 and tc_2 shows that condition a can independently affect the outcome of decision S and forms a MC/DC pair for condition a, because $S(tc_1) \neq S(tc_2)$ and the two test cases tc_1 and tc_2 are only different in the truth value assigned to condition a. Similarly, the MC/DC pairs $\{tc_1, tc_3\}$, $\{tc_1, tc_4\}$, $\{tc_5, tc_6\}$, $\{tc_5, tc_7\}$ show the independent effect on conditions b, c, d, e, respectively. The size of this test set is seven.

Assume at most m $(m \leq n)$ conditions can be MC/DC covered in decision S. Obviously, all the conditions in this example can be MC/DC covered, that is, $m = n = 5$. Since a pair of test cases are required to cover each condition using the MC/DC requirement, the upper bound of the size of a MC/DC test set is $2m$. On the other hand, Hayhurst et al. [5] pointed out that at least $n + 1$ test cases are required for minimal test set. Therefore the size of MC/DC test set are in the interval $[m + 1, 2m]$. Also consider example 1, although Table 1 demonstrates a test set with seven elements, we

Table 1: A test set of decision $S = (a \wedge b \wedge c) \vee (\neg c \wedge d \wedge e)$ for MC/DC

Test cases	a	b	c	d	e	$S(tc_i)$
tc_1	1	1	1	0	0	1
tc_2	0	1	1	0	0	0
tc_3	1	0	1	0	0	0
tc_4	1	1	0	0	0	0
tc_5	0	0	0	1	1	1
tc_6	0	0	0	0	1	0
tc_7	0	0	0	1	0	0

can construct minimal test set whose size is 6 (5+1) {01111, 10001, 10010, 10011, 10111, 11111} to satisfy the MC/DC coverage.

The upper bound $2m$ is quite small compared to the number of all possible test cases 2^n, and grows linearly in the number of conditions of decision S. Therefore the optimal test set (the test set with minimum size) is efficient for testing Boolean specifications with low cost. Hence how to efficiently construct the optimal test set is a crucial problem in MC/DC testing. Since the MC/DC coverage criterion focuses on conditions and Boolean expressions, we naturally take the well-researched SAT techniques into consideration.

2.3 SAT Problem

A SAT problem (Propositional Satisfiability Problem) determines whether there exists a truth assignment to the variables of a Boolean formula (a decision) such that it becomes satisfied. It is the first proved NP-complete problem and many real-world problems can be translated to SAT decision problems by means of different encoding techniques. Many highly efficient state-of-art SAT solvers have been developed, such as Minisat[1], Glucose[2], and different mathematical methods available to the SAT solver to assist in solving them with improved performance.

3 MAIN IDEAS

In this section, we present an approach to translate the problem of whether there exists a MC/DC test set with a given size into a SAT problem. Then based on this technique, we can search the optimal test set by invoking SAT solvers several times.

3.1 Overview

As we have pointed out in section 2.2, not all conditions can be MC/DC covered. For an arbitrary decision $S(x_1, x_2, \ldots, x_n)$ with n conditions, according to the definition of MC/DC coverage, a condition x_r $(1 \leq r \leq n)$ can be MC/DC covered if the following formula holds

$$S(x_1, \ldots, x_r, \ldots, x_n) \oplus S(x_1, \ldots, \neg x_r, \ldots, x_n). \qquad (3.1)$$

The satisfiability of (3.1) can be easily decided within 1ms by the state-of-art SAT solvers. Then we can figure out which conditions can not independently affect the outcome of the decision, and omit them in the next steps.

Assume that there are m $(1 \leq m \leq n)$ conditions in decision S can be MC/DC covered. In theory, we can find the minimal size of MC/DC test set in the range of $m + 1$ to $2m$. The commonly used search strategy is the binary search. If each element in the search domain $[m + 1, 2m]$ is equally likely to be searched, then the average case binary search requires logarithmic time to find such a key. However, our experimental results show that most (more than 99%) instances can be MC/DC covered by a test set with only $m + 1$ test cases. Therefore, we impose linear search strategy and start from $m + 1$. In most cases, our SAT-based approach is efficient for generating optimal MC/DC test sets, with only one invoke to the SAT solver.

Our approach for finding the optimal test set of MC/DC is outlined in Algorithm 1. In the beginning, our approach pre-processes conditions independently affecting the outcome of decision S, denoted by $\langle X', m\rangle$. Then, we impose linear search to find the minimum value of k. Lines 3-10 describe the way of translating MC/DC problem with a given size k into the propositional satisfiability problem to determine whether there exist k test cases that can MC/DC cover all m conditions in X' for decision S. In algorithm 1, C denote the Boolean constraint set. It first initializes constraint set C as empty set $\emptyset$, which records all the Boolean constraints that the problem has generated in the process of executing. We first add constraints of MC/DC problem description on the constraint set C, which will be discussed in section 3.2. That is, for each condition x_r $(x_r \in X')$, determine if there exist two test cases tc_i and tc_j that can satisfy (3.2). Then we add lexicographic ordering constraints, which will be discussed in section 3.3, into set C to break symmetry and reduce the size of the search space. Finally, we impose the Tseitin encodings method [2, 9] to translate problem constraints in propositional logic into the DIMACS file format for CNF, and then invoke SAT solver to determine whether there exists a solution of k test cases.

Algorithm 1 SAT based approach to find the optimal test set of MC/DC criterion for an arbitrary decision.

Input: An arbitrary decision $S(x_1, x_2, \ldots, x_n)$ with n conditions.
Output: The minimal test set satisfying the MC/DC criterion for decision S.

```
1: ⟨X′, m⟩ ← PreProcessing(S)
2: for k:= m + 1 to 2m do
3:     initialize: constraint set C = ∅;
4:     C ← C∪{constraints on problem description};
5:     C ← C∪{constraints of lexicographic ordering};
6:     CNF ← TseitinEncoding(C);
7:     ⟨is_sat, solution⟩ ← invokeSATSolver(CNF, time_limit);
8:     if is_sat then
9:         return solution;
10:    end if
11: end for
```

[1] http://minisat.se/
[2] http://www.labri.fr/perso/lsimon/glucose/

3.2 Problem of Generating k-size Test Set

Consider a decision $S(x_1, x_2, ..., x_n)$ with n conditions. Let $X = \{x_1, x_2, ..., x_n\}$ be the set of all conditions in S. Let $X' \subseteq X$ whose size is m be the set of conditions that independently affect the outcome of decision S. Assume we want to generate a test set $TS = \{tc_1, tc_2, \ldots, tc_k\}$ with k ($m + 1 \leq k \leq 2m$) elements, we need to construct a $k \times n$ matrix where each row represents a test case. These $k \times n$ entries can be regarded as input variables in the propositional satisfiability problem in the next translation step, that is, a Boolean variable $v_{i,j}$ represents the value of the j'th condition in the i'th test case.

Example 2. Consider the decision $S = (a \wedge b \wedge c) \vee (\neg c \wedge d \wedge e)$ with five conditions. Suppose we want to find $k = 6$ test cases to MC/DC cover this decision. We introduce 6×5 Boolean variables in the transformed SAT problem, which are shown in Table 2.

Table 2: Boolean variables in SAT problem that transformed for decision $S = (a \wedge b \wedge c) \vee (\neg c \wedge d \wedge e)$.

test cases	a	b	c	d	e
tc_1	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$v_{1,4}$	$v_{1,5}$
tc_2	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$v_{2,4}$	$v_{2,5}$
tc_3	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$v_{3,4}$	$v_{3,5}$
tc_4	$v_{4,1}$	$v_{4,2}$	$v_{4,3}$	$v_{4,4}$	$v_{4,5}$
tc_5	$v_{5,1}$	$v_{5,2}$	$v_{5,3}$	$v_{5,4}$	$v_{5,5}$
tc_6	$v_{6,1}$	$v_{6,2}$	$v_{6,3}$	$v_{6,4}$	$v_{6,5}$

Then we will add the following constraints on these variables into the description of this problem. Let the boolean variable $c_{r,i,j}$ ($1 \leq i < j \leq k$ and $1 \leq r \leq n$) denote whether the i'th and j'th test cases (tc_i and tc_j) can MC/DC cover the r'th condition x_r ($x_r \in X'$). According to the definition of MC/DC, we have the constraint

$$c_{r,i,j} \leftrightarrow ((v_{i,r} \oplus v_{j,r}) \wedge (\bigwedge_{\ell \neq r} (v_{i,\ell} \leftrightarrow v_{j,\ell})) \wedge (S(tc_i) \oplus S(tc_j))) \quad (3.2)$$

where $1 \leq \ell \leq n$.

Since each condition $x_r \in X'$ should be MC/DC covered, we have

$$\bigwedge_{x_r \in X'} (\bigvee_{1 \leq i < j \leq k} c_{r,i,j}). \quad (3.3)$$

3.3 Symmetry Breaking

Many reasoning and optimization problems exhibit symmetries that if an assignment to input variables is a solution, the isomorphisms of these assignments are also solutions. Time is wasted in visiting new solutions which are symmetric to the already visited solutions. The search time can be reduced by adding new constraints, referred as symmetry breaking constraints, such that some of the symmetric solutions are eliminated from the search space. One common method to break symmetry is to impose a constraint that orders the symmetric objects [3].

In our problem, we use a $k \times n$ matrix to represent a test set, and treat each test case as a vector. Then we order these vectors lexicographically, that is, we add constraints that each row is lexicographically smaller than the next row as symmetries elimination rule. We can give a recursive definition of lexicographic ordering for two test cases:

Definition 3.1 (Lexicographic Order). *Two test cases $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$ have a **lexicographic order** $X \leq_{lex} Y$ iff*

(1) *If $n = 1$, $\neg x_1 \vee y_1$;*
(2) *If $n > 1$, $(\neg x_1 \vee y_1) \vee ((x_1 \leftrightarrow y_1) \wedge (x_2, \ldots, x_n) \leq_{lex} (y_2, \ldots, y_n))$.*

3.4 CNF Encoding

To invoke a SAT solver to determine if there exists a solution that satisfies MC/DC criterion and then generate these k test cases, we impose the Tseitin encodings [2, 9] to transform a formula φ in propositional logic to the DIMACS file format for CNF (conjunctive normal form). The Tseitin method generates a linear number of clauses at the cost of introducing a linear number of new variables. Its basic idea is to associate a new variable f with each subformula ϕ in order to represent its truth value, and conjoin the constraint $(f \leftrightarrow \phi)$. Finally, to require the whole formula to be true, add the unit clause c where c is a variable that is associated with the whole formula. The original propositional formula and the transformed formula are equi-satisfiable, meaning that the latter is satisfiable if, and only if, the original formula is satisfiable. This definitional form of formula φ then can be easily translated into CNF by rewriting into a small set of clauses.

Example 3. Consider the following formula φ:

$$\varphi := c \rightarrow (a \vee b)$$

We first parse formula φ into a binary-tree, then from bottom to top, the Tseitin encoding would first introduce a new variable f_1 with definition $f_1 \leftrightarrow (a \vee b)$ to represent subformula $(a \vee b)$. This definition can be transformed into CNF as follows:

$$\begin{aligned} f_1 \leftrightarrow (a \vee b) &\Leftrightarrow (f_1 \rightarrow (a \vee b)) \wedge ((a \vee b) \rightarrow f_1) \\ &\Leftrightarrow (\overline{f_1} \vee (a \vee b)) \wedge (\overline{(a \vee b)} \vee f_1) \\ &\Leftrightarrow (\overline{f_1} \vee a \vee b) \wedge ((\bar{a} \wedge \bar{b}) \vee f_1) \\ &\Leftrightarrow (\overline{f_1} \vee a \vee b) \wedge (\bar{a} \vee f_1) \wedge (\bar{b} \vee f_1) \end{aligned}$$

Similarly, it would introduce a new variable f_2 for subformula $(c \rightarrow f_1)$, that is $f_2 \leftrightarrow (c \rightarrow f_1)$, which can be transformed to clausal form as $(\overline{f_2} \vee \bar{c} \vee f_1) \wedge (c \vee f_2) \wedge (\overline{f_1} \vee f_2)$. Therefore, the whole formula are equi-satisfiable with $(f_1 \leftrightarrow (a \vee b)) \wedge (f_2 \leftrightarrow (c \rightarrow f_1)) \wedge f_2$, which can be translated into CNF as $(\overline{f_1} \vee a \vee b) \wedge (\bar{a} \vee f_1) \wedge (\bar{b} \vee f_1) \wedge (\overline{f_2} \vee \bar{c} \vee f_1) \wedge (c \vee f_2) \wedge (\overline{f_1} \vee f_2) \wedge f_2$.

The followings are several equi-satisfiable transformations:

- AND: $f \leftrightarrow (a \wedge b) \Leftrightarrow (a \vee \bar{f}) \wedge (b \vee \bar{f}) \wedge (\bar{a} \vee \bar{b} \vee f)$
- OR: $f \leftrightarrow (a \vee b) \Leftrightarrow (\bar{a} \vee f) \wedge (\bar{b} \vee f) \wedge (a \vee b \vee \bar{f})$
- EQU: $f \leftrightarrow (a \leftrightarrow b) \Leftrightarrow (a \vee \bar{b} \vee \bar{f}) \wedge (\bar{a} \vee b \vee \bar{f}) \wedge (a \vee b \vee f) \wedge (\bar{a} \vee \bar{b} \vee f)$
- XOR: $f \leftrightarrow (a \oplus b)$ is the same as $f \leftrightarrow (\bar{a} \leftrightarrow b)$.
- IMPLY: $f \leftrightarrow (a \rightarrow b)$ is the same as $f \leftrightarrow (\bar{a} \vee b)$.

In the process of associating a new variable f with subformula ϕ by Tseitin encoding, we may find some redundant f generated when the current subformula has already occurred before. For example, consider a formula $(\bar{c} \rightarrow (a \vee b)) \oplus (c \rightarrow (a \vee b))$, the subformula $a \vee b$ appears more than once, and when we search the expression

binary-tree from bottom to top, we will treat them as different occurrences, and associate two new variables f and f' on them, respectively. We say f and f' are redundant. To eliminate common subexpressions and reduce the number of new variables generated by Tseitin encoding method, we impose the hash map searching strategy by designing a hash function H and creating a hash table to store the information of all f that have been generated according to different subformula that the algorithm meet. For each subformula ϕ we meet currently, we first transform ϕ into a unique search `key`. A simple way to design a search key for subformula $(a \cdot b)$, where "$\cdot$" indicates a binary operator, is to append `idx_a`, `idx_op` and `idx_b` respectively, where we assume that the index of a and b are defined by `idx_a` and `idx_b`, and the index of "$\cdot$" is defined by `idx_op`. Then we use hash function H to compute an index that suggests where the entry can be found in the hash table. If the hash searching finds the `value`(f) of $H(key)$, we will use this variable f that already exists to associate the subformula ϕ; otherwise, introduce a new variable to denote the subformula ϕ.

4 EXPERIMENTAL SETUP

In this section, to evaluate the performance of our approach described in this paper, we developed a tool based on our method to generate the minimum number of test cases that satisfies the MC/DC criterion, and we give three kinds of benchmarks that are used in our experiment.

4.1 Three Kinds of Benchmarks

We performed a set of experiments to analyze the effectiveness and cost of our approach. The benchmarks we used here are divided into three parts: (1) TCAS II benchmarks, which were introduced by Weyuker et al. [10] and consisted of 20 Boolean expressions selected from the TCAS II specification, the traffic collision avoidance system. Here we only selected top 10 most complicated expressions listed in Figure 1. (2) Several hand made Boolean expressions, which are organized by ourselves and listed in Figure 2. (3) Random benchmarks, which are generated randomly with different kinds of complexities.

(1) $(\overline{ab})(d\bar{e}\bar{f} + \bar{d}e\bar{f} + \bar{d}\bar{e}\bar{f})(ac(d+e)h + a(d+e)\bar{h} + b(e+f))$
(2) $(\bar{a}b + a\bar{b})(\overline{cd})(\overline{gh})((ac+bd)e(fg+\bar{f}h))$
(3) $a(\bar{b} + \bar{c} + bc(\overline{\bar{f}gh\bar{i}})(\overline{\bar{f}glk + \bar{g}\bar{i}k})) + f$
(4) $(a((c+d+e)g + af + c(f+g+h+i)) + (a+b)(c+d+e)i)(\overline{ab})(\overline{cd})(\overline{ce})(\overline{de})(\overline{fg})(\overline{fh})(\overline{fi})(\overline{gh})(\overline{hi})$
(5) $(\bar{a}b + a\bar{b})(\overline{cd})(\overline{gh})(\overline{jk})((ac+bd)e(\bar{i} + \bar{g}\bar{k} + \bar{j}(\bar{h}+\bar{k})))$
(6) $(\bar{a}b + a\bar{b})(\overline{cd})(f\bar{g}\bar{h} + \bar{f}g\bar{h} + \bar{f}\bar{g}\bar{h})(\overline{jk})((ac+bd)e(f + (i(gj + hk))))$
(7) $a(\overline{\bar{d} + \bar{e} + de(\overline{\bar{f}gh\bar{i}})(\overline{\bar{f}glk + \bar{g}\bar{i}k})}) + (\overline{\bar{f}gh\bar{i} + \bar{g}hi})(\overline{\bar{f}glk + \bar{g}\bar{i}k})(b + c\bar{m} + f)$
(8) $(a(\bar{d} + \bar{e} + de(\overline{\bar{f}gh\bar{i}})(\overline{\bar{f}glk + \bar{g}\bar{i}k})) + (\overline{\bar{f}gh\bar{i}})(\overline{\bar{f}glk + \bar{g}\bar{i}k})(b + c\bar{m} + f))(a\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}\bar{b}c)$
(9) $a\bar{b}\bar{c}((\overline{f(g + \bar{g}(h+i))})) + f(g + \bar{g}(h+i))\bar{d}\bar{e})(\overline{jk + \bar{j}l\bar{m}})$
(10) $a\bar{b}\bar{c}(f(g + \bar{g}(h+i))(\bar{e}\bar{n} + d)) + \bar{n}(jk + \bar{j}l\bar{m})$

Figure 1: The TCAS II benchmarks.

(1) $abc + def + gha$
(2) $abc + cde + efg$
(3) $ab + bc + cd + da$
(4) $a\bar{f}cde + \bar{a}fhij + b\bar{g}\bar{c}\bar{d}\bar{e} + \bar{b}g\bar{h}\bar{i}\bar{j}$
(5) $\bar{b}\bar{c}\bar{d}\bar{e} + \bar{a}cde$

Figure 2: The hand made benchmarks.

In the experiment, we use TCAS II benchmarks to represent the complexity of decisions in real programs, and to show the efficiency of our method. Because the complexity of the decisions extracted from TCAS system are quite similar to the complexity of decisions in a real program. Wu et al. [11] proposed a greedy strategy to automatically generate test data for MC/DC in a real program. However, their approach as an incomplete algorithm might only find the approximate solutions of minimal test sets for MC/DC. We use several hand made but simple Boolean expressions to show the approximate solutions that the greedy strategy can only find, compared with our method. Then to analyze the cost and effectiveness of our approach, we made a comparison with the greedy strategy based on random benchmarks, which are more complex but less common in real programs compared with TCAS II benchmarks.

4.2 Random Benchmark Generation

Observe that (1) all Boolean operators can be represented equivalently by three basic Boolean operators: `AND` ($\wedge$), `OR` ($\vee$), `NOT` ($\neg$). (2) according to the distributivity of `NOT` operator, $\neg(p_1 \wedge p_2)$ can be equivalently transformed to $\neg p_1 \vee \neg p_2$, and $\neg(p_1 \vee p_2)$ can be equivalently transformed to $\neg p_1 \wedge \neg p_2$ where p_1, p_2 represent two Boolean expressions. As an example, $\neg((x_1 \wedge \neg x_2) \vee x_3)$ can be translated into $(\neg x_1 \vee x_2) \wedge \neg x_3$. (3) the two basic Boolean operators $\wedge$ and $\vee$ are binary operators. According to (1), we only consider Boolean expressions built on the operators $\wedge$, $\vee$ and $\neg$. According to (2) and (3), an arbitrary Boolean expression can be represented by a binary expression tree with two types of nodes. The internal nodes are Boolean operators, which are labeled with $\wedge$ and $\vee$, the leaves are Boolean operands, which are labeled with Boolean variables and their negations.

A simple way of generating random Boolean expressions is described as follows. We start from the tree's root node. Building a random binary Boolean expression tree can be viewed as a branching process. Let *p_branching* denote the probability of branching at each node. That is, the current node either represents a Boolean operator $\wedge$ or $\vee$ with probability *p_branching*, or represents a Boolean operand with probability $1-$*p_branching*. If the current node is a Boolean operator, we label this internal node by $\wedge$ or $\vee$ with uniform probability, then recursively generate the left and right subtrees of this node. If the current node is a Boolean operand, we label this leaf by one of the $2n$ literals again with uniform probability. To control the height of tree, the value of probability *p_branching* is changed dynamically according to the depth of current node. Assume the node at depth d, the probability *p_branching* is decreased by $d * p\_dec$, where $p\_dec \in [0,1]$ is a constant and usually assigned with a very small value. In the experiment, we use *p_branching* = 0.9 and *p_dec* = 0.07.

4.3 Experimental Environment

Our tool first translates the MC/DC coverage problem which is an optimization problem into the propositional satisfiability problem, and then the tool invokes a SAT solver to determine whether it can be satisfied. The SAT solver that we invoked in our tool was MiniSat, which is minimalistic and open-source with high-performance. All the instances are running on an Intel(R) Core(TM) i7-6700 CPU@3.4GHz machine with 64-bit linux operating system.

5 EXPERIMENTAL RESULTS

In this section, we give some experimental results based on three benchmarks and analyze the performance of our approach.

5.1 Results for TCAS II Benchmarks

For each Boolean expression, we first count the number of conditions (denoted by n) in the decision. Then using a SAT solver to determine the number of conditions that independently affect the outcome of the decision (denoted by m). For experimental results, we use k_{min} to represent the size of minimal test set for MC/DC that algorithms have generated. Let tt denote how many times the SAT solver has been invoked (except prepocessing to find the size m) by our approach. Results for TCAS II benchmarks by our tool and greedy approach developed by Wu et al. [11] are shown in Table 3.

Table 3: Results for TCAS II Benchmarks

#	n	m	Greedy Approach		SAT based Approach		
			k_{min}	Time (s)	k_{min}	tt	Time (s)
(1)	7	7	8	0.09	8	1	0.07
(2)	8	8	9	0.11	9	1	0.08
(3)	9	9	10	0.15	10	1	0.13
(4)	9	9	10	0.18	10	1	0.14
(5)	10	10	11	0.18	11	1	0.20
(6)	11	11	12	0.25	12	1	0.24
(7)	12	12	13	0.42	13	1	0.28
(8)	12	12	13	0.54	13	1	0.26
(9)	13	13	14	0.43	14	1	0.32
(10)	14	14	15	0.54	15	1	0.62

From Table 3, we can see that both SAT based approach and greedy approach can generate the minimal test sets that satisfy the MC/DC criterion, and the execution time required for each instance is within 1 second for the TCAS II benchmarks. Therefore, in real programs, our method can have a high efficiency of solving problems.

5.2 Results for Hand Made Benchmarks

Table 4 shows the minimum number of test cases for MC/DC criterion that are generated by our approach based on SAT and greedy approach developed by Wu et al. [11], respectively. For instances (1)-(4), our approach based on SAT can find the minimum size of test set of MC/DC, while greedy approach can not. Instance (5) shows that both SAT based approach and greedy approach can find the optimal test set of MC/DC, whose size is $m + 2$ ($m = 5$).

Table 4: Results for Hand Made Benchmarks

#	n	m	Greedy Approach		SAT based Approach		
			k_{min}	Time (s)	k_{min}	tt	Time (s)
(1)	8	8	10	0.25	9	1	0.09
(2)	7	7	9	0.16	8	1	0.07
(3)	4	4	6	0.01	5	1	0.01
(4)	10	10	12	0.43	11	1	0.87
(5)	5	5	7	0.11	7	1	0.10

5.3 Results for Small-scale Decisions

In the experiment, we found a phenomenon that most of the instances can be MC/DC covered by $m + 1$ test cases. Therefore, we enumerated all small-scale decisions (within 5 conditions) via building the truth table of a decision and generated the optimal test set of MC/DC for these instances. In the case of $n = 1$ (2 or 3 or 4), all decisions that are constructed by n conditions can be MC/DC covered by $m + 1$ test cases. When $n = 5$, since the number of all possible decisions, which consist of 5 conditions, is quite big (more than 4000,000,000), we enumerated 800,000 such instances and found only 40 decisions whose optimal MC/DC test set is of size $m + 2$. These experimental results indicate that most (more than 99%) of instances can be MC/DC covered by only $m + 1$ test cases.

5.4 Results for Random Benchmarks

To analyze the cost and effectiveness of our approach, we made a comparison with greedy strategy based on random benchmarks. For each instance in random benchmarks, let lt denote the number of literals in the Boolean expression, and fv denote the average frequency of occurrence for each condition. The values of n, m, lt and fv show the complexity of a Boolean expression. Since these instances in the random benchmarks are complex, we set a time limit for SAT solver within 5 minutes in each iteration.

Since previous experimental results indicate that the size of optimal MC/DC test set with high probability is $m + 1$, we randomly generated 20,000 Boolean expressions to further prove this phenomenon empirically, and the number of variables in each Boolean expressions is in the interval [10, 25]. In the experiment, we found about 74.2% benchmarks (14,835 instances) can be MC/DC covered by $m + 1$ test cases via greedy approach developed by Wu et al. [11]. Then we randomly selected 800 instances (50 instances for each size of m) from the rest of 25.8% benchmarks and used our SAT based tool on these instances.

The experimental results are shown in table 5 and table 6. Table 5 shows the average increase rate and average execution time for each size of m. The ***increase rate*** is defined as (upper_bound - k_{min} + 1) / (upper_bound - lower_bound + 1), to represent the percentage of test cases that can be reduced. In MC/DC problem, that is ($2m$ –

$k_{min} + 1)/m$. Table 6 shows the minimum number of test cases for MC/DC criterion that are generated by our approach based on SAT and greedy approach developed by Wu et al. [11], respectively. These 15 instances are selected from the random benchmarks. The experimental results show that both SAT based approach and greedy approach can efficiently find the optimal solutions for MC/DC criterion in small-scale (less than 20 conditions). When the number of conditions is in the range of 21 to 25, SAT based approach can also find a smaller test set compared to greedy approach within an acceptable time. The SAT based approach is more effective than greedy approach, it can reduce more test cases for MC/DC criterion. Besides, the experimental results also indicate that more than 99% of instances can be MC/DC covered by only $m + 1$ test cases.

Table 5: Results for Random Benchmarks

m	Greedy Approach		SAT based Approach	
	Increase Rate	Avg. Time (s)	Increase Rate	Avg. Time (s)
10	87.1%	0.23	100%	0.08
11	86.7%	0.31	100%	0.10
12	87.1%	0.43	100%	0.19
13	90.6%	0.70	100%	0.72
14	86.3%	1.14	100%	1.15
15	89.2%	1.13	100%	1.67
16	89.1%	1.25	100%	2.85
17	91.1%	1.61	100%	4.69
18	90.6%	1.67	100%	7.68
19	92.3%	1.79	99.8%	37.8
20	88.1%	2.85	100%	17.3
21	91.6%	3.24	99.6%	75.3
22	91.6%	3.77	98.7%	131.2
23	90.1%	4.05	98.3%	182.7
24	86.1%	6.11	94.3%	486.5
25	83.0%	8.78	88.8%	893.5

6 DISCUSSION

In this section, we give some discussions about the different definitions of condition and the phenomenon of size $m + 1$ test set.

6.1 Different definitions of condition

The definition of *conditions* in the document [8] indicates that if a condition appears more than once in a decision, each occurrence is a distinct condition. Hence, all conditions can be treated as different from each other. In such situation, Kangoye et al. [6] pointed out that the size of a minimum set of test cases that achieve MC/DC is $n + 1$ (since each condition occurs only once, therefore $m = n$) for a decision of n conditions, and they also gave a direct construction of these $n + 1$ test cases. Hence, if multiple occurrences of the same variable are regarded as different conditions, the test case generation problem for this criterion is trivial. In this paper we treat the multiple occurrences of a condition as identical, that is more close to real-world testing scenario and makes it difficult to generate the optimal test set of MC/DC coverage.

6.2 Phenomenon of size $m + 1$ test set

In the experimental results, we find that most of decisions can be MC/DC covered by $m+1$ test cases. This phenomenon is intuitively weird but reasonable. Consider the conclusion of Kangoye et al. [6], for a decision if there exists an equivalent form in which all conditions occur only once in the decision, we can construct a MC/DC test set whose size is $m+1$. Take decision $(a \wedge b) \vee (a \wedge c)$ as an example, although condition a occurs twice, it can be equivalently transformed to $a \wedge (b \vee c)$. This phenomenon also indicates that the MC/DC coverage is an effective criterion that can achieve a high ability of fault detection while using a small set of test cases (mainly $m + 1$).

7 RELATED WORKS

The MC/DC criterion is highly recommended and commonly used especially in some avionic systems [8]. How to automatically generate the optimal test set satisfying the MC/DC criterion is a very important topic in software testing. There are many related works that have been done, such as greedy or meta-heuristic search based strategy [1, 4, 11], binary tree based method and truth table based method [6]. Kangoye et al. [6] presented two binary tree based methods to generate the minimal test set satisfying MC/DC coverage. However, it may happen that none of the two approaches manages to find a solution (e.g., two occurrences of the same condition in a decision) to a decision. They also presented a truth table based approach, which is computationally complex for large decision. The greedy or meta-heuristic search strategy proposed by [1, 4, 11] is efficient, however, these approaches as incomplete algorithms may not guarantee that the solutions are optimal. Yan et al.[12] presented a SAT based method to automatically generate test cases for MUMCUT coverage. Although MUMCUT criterion is proved to have stronger fault-detecting ability of test sets than the MC/DC and some other related coverage criteria, the constraints of the MUMCUT criterion are more complicated than other related criteria, and a MUMCUT test set is on average about 2 or 3 times as large as a MC/DC test set [13]. The experimental results according to Yu et al. [13] show that MC/DC is indeed an effective criterion and possesses very good fault-detecting ability for all faults except LIF, yet generally it does not require a very large test set.

8 CONCLUSION

Existing approaches mainly treat two occurrences of the same condition in a decision as distinct, while our approach treats them as identical, therefore the minimal test set of MC/DC may no longer be satisfied by $m+1$ test cases. In this paper, we presented a SAT based approach to automatically generate minimal test set of MC/DC coverage for an arbitrary Boolean expression. In the experiment, we analyzed the cost and effectiveness of our approach, and made a comparison with greedy strategy based approach. Experimental results indicate that our method executes fast for a complex mixed decision within 20 conditions and can always find the optimal test set of MC/DC. Our experimental results also first reveal a

Table 6: Results of 15 Random Instances

#	n	m	lt	fv	Greedy Approach		SAT based Approach		
					k_{min}	Time (s)	k_{min}	tt	Time (s)
(1)	11	11	19	1.72	14	0.32	12	1	0.10
(2)	12	12	55	4.58	16	0.64	13	1	0.25
(3)	13	13	54	4.15	17	0.99	14	1	0.41
(4)	14	14	53	3.78	18	1.18	15	1	1.04
(5)	15	15	74	4.93	19	1.59	16	1	1.82
(6)	16	14	22	1.36	17	0.98	15	1	0.49
(7)	17	17	37	2.18	20	1.58	18	1	1.74
(8)	18	18	52	2.89	21	2.02	19	1	6.60
(9)	19	18	28	1.47	22	1.97	19	1	1.68
(10)	20	19	37	1.85	22	1.26	20	1	1.73
(11)	20	20	33	1.65	23	1.64	21	1	17.3
(12)	21	21	41	1.95	24	2.91	22	1	58.10
(13)	22	22	43	1.95	26	9.62	23	1	39.64
(14)	24	23	60	2.50	27	6.60	26	3	682.94
(15)	25	23	59	2.36	26	5.87	24	1	38.75

phenomenon that most (more than 99%) of the decisions can be MC/DC covered by only $m + 1$ test cases that is the lower bound in theory, where m is the number of conditions that can be MC/DC covered.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their helpful comments and suggestions. This work is supported by the National Key Basic Research (973) Program of China (Grant No. 2014CB340701) and the National Natural Science Foundation of China (Grant No. 61672505) as well as the CAS/SAFEA International Partnership Program for Creative Research Teams.

REFERENCES

[1] Zeina Awedikian, Kamel Ayari, and Giuliano Antoniol. 2009. MC/DC automatic test input data generation. In *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009.*

[2] A. Biere, M. Heule, H. van Maaren, and T. Walsh. 2009. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications.* IOS Press, Amsterdam, The Netherlands.

[3] Pierre Flener, Alan M. Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. 2002. Breaking Row and Column Symmetries in Matrix Models. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP '02).* Springer-Verlag, London, UK, 462–476.

[4] Kamran Ghani and John A. Clark. 2009. Automatic Test Data Generation for Multiple Condition and MCDC Coverage. In *The Fourth International Conference on Software Engineering Advances, ICSEA 2009, 20-25 September 2009, Porto, Portugal.* 152–157.

[5] Kelly J. Hayhurst, Dan S. Veerhusen, John J. Chilenski, and Leanna K. Rierson. 2001. *A Practical Tutorial on Modified Condition/Decision Coverage.* Technical Report. NASA Langley Technical Report Server, NASA/TM-2001-210876.

[6] Sekou Kangoye, Alexis Todoskoff, Mihaela Barreau, and Philippe Germanicus. 2015. MC/DC Test Case Generation Approaches for Decisions. In *Proceedings of the 24th Australasian Software Engineering Conference, ASWEC 2015, Volume II, Adelaide, SA, Australia, September 28 - October 1, 2015.* 74–80.

[7] CAST-10 Paper. 2002. *What is a "Decision" in Application of Modified Condition/Decision Coverage (MC/DC) and Decision Coverage (DC)?*

[8] RTCA/DO-178B. 1992. *Software Considerations in Airborne Systems and Equipment Certification.* RTCA, Inc.

[9] G. S. Tseitin. 1983. *On the Complexity of Derivation in Propositional Calculus.* Springer Berlin Heidelberg, Berlin, Heidelberg, 466–483.

[10] Elaine J. Weyuker, Tarak Goradia, and Ashutosh Singh. 1994. Automatically Generating Test Data from a Boolean Specification. *IEEE Trans. Software Eng.* 20, 5 (1994), 353–363.

[11] Tianyong Wu, Jun Yan, and Jian Zhang. 2014. Automatic Test Data Generation for Unit Testing to Achieve MC/DC Criterion. In *Eighth International Conference on Software Security and Reliability, SERE 2014, San Francisco, California, USA, June 30 - July 2, 2014.* 118–126.

[12] Jun Yan and Jian Zhang. 2006. SAT based automated test case generation for MUMCUT coverage. *The 17th IEEE International Symposium on Software Reliability Engineering (Student Program Papers) ISSRE* (2006).

[13] Yuen-Tak Yu and Man Fai Lau. 2006. A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions. *Journal of Systems and Software* 79, 5 (2006), 577–590.