MC/DC Test Cases Generation based on BDDs *

Faustin Ahishakiye[1], José Ignacio Requeno Jarabo[1,2], Lars Michael Kristensen[1], and Volker Stolz[1]

[1] Western Norway University of Applied Sciences, Bergen, Norway
[2] Complutense University of Madrid, Madrid, Spain
{fahi,jirj,lmkr,vsto}@hvl.no, jrequeno@ucm.es

Abstract. We present a greedy approach to test-cases selection for single decisions to achieve MC/DC-coverage of their Boolean conditions. Our heuristics take into account "don't care" inputs through three-valued truth values that we obtain through a compact representation via reduced-ordered binary decision diagrams (roBDDs). In contrast to an exhaustive, resource-consuming search for an optimal solution, our approach quickly gives frequently either optimal results, or otherwise produces "good enough" results (close to the optimal size) with little complexity. Users obtain different — possibly better — solutions by permuting the order of conditions when constructing the BDD, allowing them to identify the best solutions within a given time budget. We compare variations on metrics that guide the heuristics.

1 Introduction

Software testing techniques that achieve coverage effectiveness and provide test cases are cost intensive [31]. Certification standards for safety assurance such as DO-178C [28] in the domain of avionic software systems require software with the highest safety level (Level A) to show modified condition decision coverage (MC/DC) [10]. One of the advantages of MC/DC is that for a decision with n conditions, it may be satisfied with less test cases: between a lower-bound of $n+1$ and upper-bound of $2n$ test cases, compared to multiple condition coverage (MCC) which requires 2^n test cases. MC/DC requires that each condition in a decision shows an independent effect on that decision's outcome by (1) varying just that condition while holding fixed all other possible conditions (**UC-MC/DC**), or (2) varying just that condition while holding fixed all other possible conditions that could affect the outcome. This criterion of showing independence effect for conditions is unique for MC/DC compared to other structure coverage criteria.

While trying all possible combinations is exhaustive and requires tremendous resources [18], as well as becoming impracticable for a high number of conditions [23,19], finding a test set equal or closer to $n+1$ with MC/DC assurance is also

* This work was supported by the Spanish Ministry of Science and Innovation under project FAME (grant nr. RTI2018-093608-B-C31), the Comunidad de Madrid under project FORTE-CM (grant nr. S2018/TCS-4314) co-funded by EIE Funds of the European Union, and SFI Smart Ocean NFR Project 309612/F40.

a non-trivial task [24,15]. Therefore, it is important to investigate new strategies for generating good test suites both in terms of number of test cases [10] and coverage adequacy [34,14] with little complexity and with reasonable resources.

In this paper, we present a novel and alternative approach to test case generation satisfying MC/DC based on reduced-ordered binary decision diagrams (roBDDs) which are a concise representation of Boolean expressions. roBDDs are widely used in different areas such as computer aided design (CAD) tasks [26], symbolic model checking [11,26], and verification of combinational logic [20,29]. Due to their reduced form compared to other Boolean expressions representations such as disjunctive or conjunctive normal form, truth tables and formula equivalence [35]; roBDDs offer a unique normal form and were also already used in test cases generation [17,22] for different coverage criteria other than MC/DC.

We present an algorithm that takes as input the roBDD representing a Boolean expression and constructs a set of MC/DC pairs. For a decision of n conditions, we generate n pairs that contain between $n+1$ to $2n$ test cases altogether. We select paths based on their length in roBDDs and reuse factor ($\alpha()$). The reuse factor refers to the number of pairs that use a given path.

We propose and compare heuristics with different preferences with respect to three-valued truth-values (1, 0 and ?) and the length of paths in the roBDD. All of them maximize the reuse factor ($\alpha()$) together with a second criteria, namely: the longest paths in BDD ($\mathcal{H}_{LPN}$, $\mathcal{H}_{LPB}$), the longest paths which may merge ($\mathcal{H}_{LMMN}$,$\mathcal{H}_{LMMB}$), and the longest paths with better size ($\mathcal{H}_{LPBS}$). Each type of heuristic implements two different flavors which sort the BDD paths depending on the interpretation of the reuse factor as a natural number ($\mathcal{H}_{LPN}$, $\mathcal{H}_{LMMN}$) or as a boolean value ($\mathcal{H}_{LPB}$, $\mathcal{H}_{LMMB}$)(e.g., $\alpha(p,\psi) < \alpha(q,\psi)$). Our algorithm is implemented in Python and the PyEDA library [13]. We test our algorithm on the Traffic Alert and Collision Avoidance System (TCAS II) benchmarks [33] which are widely used in the literature [19,21,37,22,17].

BDDs are sensitive to conditions ordering, such that different orders yield different BDDs and their size in the worst case grows to 2^{2^n} nodes [27]. As the number of nodes increases there are many paths to select MC/DC pairs from. We present evidence that to find an optimal or "good enough" solutions, instead of a search with backtracking, it is sufficient to try a few different permutations.

The rest of this paper is organized as follows: in Section 2 we present our terminology, notations and a background on MC/DC and BDDs. Section 3 describes our approaches and algorithm for generating test cases satisfying MC/DC based on BDDs. Section 4 explains the implementation of our algorithm and discuss the results. In Section 5 we provide the state of the art of the existing related work. Finally, we present the concluding remarks and future work in Section 6.

2 Background

In this section, we provide the background on MC/DC and BDDs. We present several basic definitions and terminology which are used throughout this paper. Conditionals in source code, as well as logical expressions in software specifica-

tions can be formalized as Boolean expressions. Both BDDs and MC/DC deal with Boolean expressions.

Definition 1 (Boolean expression). *A Boolean expression is defined as an expression that can be evaluated to either* true *(T) or* false *(F) and can contain connectives: NOT, AND, OR, XOR (exclusive-or), denoted by* $\neg$, $\wedge$, $\vee$, *and* $\oplus$ *respectively.*

There has been some confusion on what is a condition and decision in the context of source code and the Certification Authorities Software Team (CAST) provided suitable definitions [7]: each occurrence of a condition is considered as a distinct condition, whereas we treat multiple occurrences of a variable as one condition, where c and $\neg c$ are strongly coupled conditions.

Definition 2 (Condition). *A* condition *denotes a logical indivisible (atomic) expression containing no Boolean operators except for the unary operator* ($\neg$)*. It contains a Boolean variable represented by* a, b, c,..., *defined over "0" or "1".*

Definition 3 (Decision). *A* decision *is a Boolean expression composed of conditions and zero or more Boolean operators. It is denoted by* $D = c_1 \square c_2 \square c_3 \cdots \square c_i \square \cdots \square c_n$, *where* c_i, $(1 \leq i \leq n)$ *are Boolean conditions and* $\square$ *stands for a binary Boolean operator. A decision is also known as a Boolean function.*

Definition 4 (Two/Three-valued test case). *Given a decision* D, *a* test case *is a truth vector* $tc = (I_1, I_2, I_3, \cdots, I_n)$ *where* $I_i \in \{0, 1\}$ *(respectively,* $\{0, 1, ?\}$*) are the inputs assigned to each conditions.* ? *is known as "don't care" meaning that a condition does not need to be evaluated due to short-circuiting. A* set of test cases *for a given decision is called a* test suite. *We denote the projection onto the truth-value at the position corresponding to some condition* c *in the test case* tc *as* $tc[c]$.

2.1 Modified condition decision coverage (MC/DC) criterion

We first give the well-known definitions for two-valued truth values, and will later extend the definitions into the three-valued setting. MC/DC subsumes the existing logical coverage criteria such as condition coverage (CC): each condition tested once true and false, decision coverage (DC): a decision is evaluated once true and once false, and multiple condition coverage (MCC): an exhaustive testing that requires all possible combination of inputs. For MC/DC each condition has to independently affect the decision's outcome. According to DO-178C [28] and CAST-10 [7] the following definition has been provided for MC/DC:

Definition 5 (MC/DC [30]).

A decision is said to be MC/DC covered iff: (i) Every point of entry and exit in the program has been invoked at least once, (ii) every condition in a decision in the program has taken all possible outcomes at least once, (iii) every decision in the program has taken all possible outcomes at least once, (iv) each condition in a decision has shown to independently affect that decision's

*outcome by: (1) varying just that condition while holding fixed all other possible conditions(**UC-MC/DC**), or (2) varying just that condition while holding fixed all other possible conditions that could affect the outcome (**Masking MC/DC**).*

The coverage of program entry and exit in the Definition 5 is not directly connected with the main point of MC/DC [32], as we only consider expressions, not programs. The most interesting part of the MC/DC definition is showing the independent effect, which demonstrates that each condition of the decision has a defined purpose. The item (1) in the definition defines the unique cause MC/DC which original MC/DC [9]. The item (2) has been introduced in DO-178C to clarify that so-called *Masked MC/DC* is allowed [6,28]. Masked MC/DC means that it is sufficient to show the independence effect of a condition by holding fixed only those conditions that could actually influence the outcome. In our analysis, we are interested in generating MC/DC test cases that show an independence effect of each condition in the decision with acceptable size.

Definition 6 (Independence effect of a condition, independence pair, $\oplus_c$). *Given two test cases tc, tc' for a decision D, we call tc* independent from *tc'* on condition *c, iff i) $D(tc) = \neg D(tc')$ (they evaluate to opposite truth values), and ii) $tc \oplus_c tc'$, where $\oplus_c$ means they differ exactly only in the input position corresponding to condition c. We then say that "tc and tc' form an independence pair" (for some condition c), written $uc(tc, tc')$.*

We will later see that in our three-valued interpretation, a test case cannot form an independence pair if it does not contain enough concrete input to evaluate to either true or false. We now reformulate the general definition of MC/DC from Def. 5 for our purposes:

Definition 7 (MC/DC-cover). *Given a decision D and set of test cases ψ, we say that ψ* MC/DC-covers *D, iff $\forall c \in D$, $\exists tc, tc' \in \psi : tc \oplus_c tc' \wedge uc(tc, tc')$ (tc is independent from tc' for every condition c).*

In other words, a set is an MC/DC-cover for a decision D, if for every condition, there exists a pair of test cases in that set which shows the independence effect of that condition by evaluating to opposing truth values.

Example 1. Consider a decision $D = (a \wedge b) \vee c$. The truth table representing MCC and all possible MC/DC pairs is given in Table 1(a). Each pair is showing the independence effect for a condition. The advantage of MC/DC over MCC can be seen from Table 1(a). MCC requires eight test cases whereas all possible MC/DC pairs contain seven test cases. Indeed, only the four test cases shown in Table 1(b) are required to achieve MC/DC [10,9]. However, choosing a set equal or closer to minimal number of test cases is non-trivial for testers, especially when there is more than one MC/DC pair for a certain condition, for example, condition c can be covered by either of three pairs (indicated in parentheses), as shown in Table 1(a).

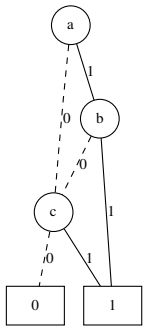


Fig. 1: roBDD: $D = (a \wedge b) \vee c$

tc	a	b	c	D	MC/DC pairs
1	0	0	0	0	
2	0	0	1	1	c(1,2)
3	0	1	0	0	
4	0	1	1	1	c(3,4)
5	1	0	0	0	
6	1	0	1	1	c(5,6)
7	1	1	0	1	a(3,7),b(5,7)
8	1	1	1	1	

(a) MCC & All MC/DC pairs

π	a	b	c	D	MC/DC pairs
1	0	?	0	0	
2	1	1	?	1	a(1,2)
3	1	0	0	0	b(2,3)
4	1	0	1	1	c(3,4)

(b) MC/DC set of paths

tc	a	b	c	D	MC/DC pairs
1	0	1	0	0	
2	1	1	0	1	a(1,2)
3	1	0	0	0	b(2,3)
4	1	0	1	1	c(3,4)

(c) MC/DC set of test cases

Table 1: MCC & MC/DC pairs for $D = (a \wedge b) \vee c$.

Chilenski et al. [10,9] investigated that for a decision D with n conditions, UC-MC/DC can be achieved with a minimal number of $n + 1$ tests while Masking MC/DC be achieved with a minimal number of $\lceil 2 * (\sqrt{n}) \rceil$ tests. This is achieved by choosing MC/DC pairs that overlap where every condition past the first one (which requires two test cases), only adds a single test case to the existing set.

Lemma 1 (Minimal MC/DC-Covers [9,1]). *If a coverage set exists for a decision D with n conditions, then there also exists a smaller set (possibly with different test cases) thereof with exactly $n + 1$ test cases such that it MC/DC-covers D for UC MC/DC.*

2.2 Overview on binary decision diagrams (BDDs)

BDDs are canonical representations of Boolean functions compared to other Boolean expressions representations such as disjunctive normal form (DNF), conjunctive normal form (CNF), truth tables and formula equivalence [35]. To reduce BDDs, conditions in a decision need to be ordered and duplicated terminals and isomorphic sub-trees have to be merged. The resulting graph is known as reduced ordered BDD (roBDD) and is shown in Figure 1 for the Example 1. BDDs represent formulas *compact* in the sense that it takes little memory to store the representation, the number of nodes in a roBDD is reduced and there is exactly one optimal and unique graph for each Boolean expression [35].

Definition 8 (Path through an roBDD, $\pi, \pi[x]$). *Given an roBDD for some decision D over Boolean variables $x_0, \ldots, x_1$. We denote a path from the root of the BDD to a terminal with π, and write $\pi[x] = 1$ if the path takes the true-branch in the node labelled with condition x (0/false respectively), and $\pi[x] =?$ if the path does not pass through a node labelled with condition x. That is, although paths through the roBDD can be of different lengths, for uniformity we always represent them as a vector with n elements.*

We also extend the evaluation of a decision wrt. some inputs ($D(0\ldots0)$) to BDDs and use $D(\pi)$ to denote the three-valued truth-value that the path represents. The obvious correspondence between a test case and a path through the roBDD is that a test case may provide more truth-values as inputs than are strictly necessary on this path. For example, an MC/DC pair of paths for condition a is $\{(0?0),(11?)\}$ as shown in row 1 & 2 of Table 1(b). The fully instantiated test cases for this pair are $\{(010),(110)\}$ (row 1 & 2, Table 1(c)).

3 Approaches and algorithm for test cases generation

Our approach and heuristics for test case generation are based on roBDDs that guide our search for test case selection. We start with a set of roBDDs paths from the root and construct sets satisfying MC/DC for all conditions, where each set contains n MC/DC pairs.

BDDs are sensitive to variable ordering: to deal with the ordering effect, we collect solutions for a number of permutations on the variable ordering. As the number of conditions in a decision increases, the number of permutations ($n!$ for n conditions) increases over-exponentially. Since generating the set of solutions for all permutation would be infeasible in those cases, we show that for few permutations we generate some test suites of minimal size, based on the selection methods defined in Subsection 3.2. In the following, we assume that all BDDs that occur are roBDDs.

3.1 Theorems and definitions for MC/DC in terms of BDDs

The core of our contribution is as follows: our algorithm produces a set of three-valued test cases, which we can instantiate to fulfill the original definition of MC/DC. We first extend general results from the standard two-valued Boolean logic to a three-valued logic.

Definition 9 (Three-valued independence pair, $\oplus_c^3$). *Given two* three-valued *test cases tc, tc' for a decision D, we write $uc3(tc, tc')$ iff*
i) $D(tc) = \neg D(tc')$ (they evaluate to opposite concrete *truth values), and ii) $tc \oplus_c^3 tc'$, where $\oplus_c^3$ means at least one of the inputs for some condition c is a concrete truth value, and for every other condition the three-valued inputs coincide or one of them is "?".*

Example 2. Let $D(X,Y,Z) = X \wedge ((\neg Y \wedge \neg Z) \vee (Y \vee Z))$. Consider $tc = (0??)$ with $D(tc) = 0$ and $tc' = (11?)$ with $D(tc') = 1$ respectively, hence $uc3(tc, tc')$. Observe that hence also e.g. $uc3(011, 11?)$ and $uc(011, 111)$.

We next show that each three-valued independence pair can be instantiated to some two-valued independence pair by suitable substitution of unknown values. In the following, for readability, we describe functions from our implementation through their properties instead of operationally. The first function combines two compatible test cases into a single one. We need this later in our algorithm

to refine existing test cases such that we keep only one test case when two cases overlap.

Definition 10 ($merge(tc, tc')$). *Given test cases tc, tc', we obtain $\sigma = merge(tc, tc')$, where $\forall c \in C, (\sigma[c] = tc[c] \wedge tc'[c] = ?) \vee (\sigma[c] = tc'[c] \wedge tc[c] = ?)$.*

In other words, *merge* substitutes some ? in a pair of paths, such that all conditions have equal values. The result is undefined if they disagree in one position where one has true and the other false. This can be understood as unifying both test cases with each other, taking ? as free variables.

Note that we ignore the actual outcome when merging wrt. a decision, but only ever consider the inputs. As we will also consider test cases that differ in *exactly one* position, we define the following variation:

Definition 11 ($merge_x(tc, tc')$). *Given test cases tc, tc', we obtain $\sigma = merge_x(tc, tc')$, where $\forall c \in C \setminus \{x\}, (\sigma[c] = tc[c] \wedge tc'[c] = ?) \vee (\sigma[c] = tc'[c] \wedge tc[c] = ?) \wedge \sigma[\mathbf{x}] = \mathbf{tc}[\mathbf{x}]$ (emphasis added).*

Note that this definition is biased to reproduce the truth-value in the designated position x from the first input, and we will consequently later see it applied *twice*, once from left to right argument, and also from right to left argument.

Example 3. We have $merge_{c_2}((1?0), (11?)) = (110)$, but $merge_{c_2}((11?), (1?0)) = (11?)$, with c_2 the condition that is placed in the last position.

Definition 12 (Specialization $\leqq$). *Given three-valued test cases p, q, we say that $p \leqq q$ iff $\exists p' : p = merge(p', q)$ ("p specializes q").*

Due to the same format for a test case and for a roBDD path (see Def. 8), both concepts are interchangeable and $\leqq$ can specialize any of them. The relation $\leqq$ is a partial order (straightforward).

Theorem 1 (Usefulness of three-valued MC/DC). *Given a decision D and set φ of three-valued test-cases that is a three-valued MC/DC cover for D, i.e., $\forall c \in D : \exists tc, tc' \in \varphi, tc \oplus_c^3 tc' \wedge uc3(tc, tc')$. Then there exists a two-valued set of test cases $\psi \subseteq 2^{\mathcal{B}^{|D|}}$, such that:*

(1) $\forall tc, tc' \in \varphi : uc3(tc, tc') \Rightarrow \exists u, u' \in \psi : u \oplus_c u' \wedge u \leqq tc \wedge u' \leqq tc'$
(each test case pair in φ has been specialised)
(2) $\forall u, u' \in \psi : D(u) = \neg D(u') \wedge u \oplus_c u' \Rightarrow \exists tc, tc' \in \varphi : uc3(tc, tc')$
$\wedge u \leqq tc \wedge u' \leqq tc'$ (ψ is the smallest set that specialises φ).

It follows that ψ is an MC/DC-cover for D.

Proof. (1) Because of $uc3(tc, tc')$, tc or tc' have a concrete value in c and coincide for the rest of conditions c_i, except for those positions c_i where one of the test cases is ?. Hence, $u = merge_{c_i}(tc, tc')$ returns a new test case where $u \leqq tc$ as the ? are instantiated (symmetrically, $u' = merge_{c_i}(tc', tc)$), excluding condition

c. MC/DC imposes that $u[c] = \neg u'[c]$, so the selection of tc and tc' satisfies that either a) $tc[c] = \neg tc'[c]$, or b) $tc[c] =?$ or $tc'[c] =?$. In b), $u[c] = tc[c]$ and $u'[c] = tc'[c]$: if any of these values is a ?, then they are properly instantiated so that $u \oplus_c u'$.

(2) As $u \oplus_c u'$, u and u' are equal except for condition c. Then, tc and tc' are constructed by replacing a finite number of positions in u (similarly, u') with ? such that they keep $uc3(tc, tc')$. Because tc and tc' are abstractions of u and u', $u \leqq tc \wedge u' \leqq tc'$.

Due to the specialization relation, multiple sets of two-valued test cases can be constructed that satisfy the above property: φ may contain a test case tc with "don't care" for some condition c, and also "don't care" for every other partner tc' in the pairs it is participating in. Then, this input c can be instantiated to either truth value. Our algorithm 1, which uses the roBDD to populate φ, guarantees that there will exist at least a pair tc, tc' such that $tc[c] = \neg tc'[c]$ for every condition c, if the decision can be MC/DC-covered.

Next, we define the function that identifies suitable test cases that we might want to add our set ψ. Based on the following criteria, for every uncovered condition the algorithm adds a new test case together with a complementary one such that the pair shows the independence effect of the condition.

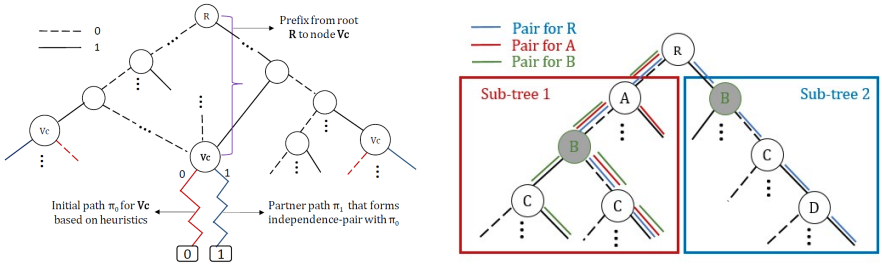
Definition 13 (Reuse factor $\alpha(\pi, \psi), \alpha_{=_3}(\pi, \psi)$). *Given the set of MC/DC pairs of paths $(\pi^\perp, \pi^\top) \in \psi$ with $D(\pi^\perp) = 0$ and $D(\pi^\top) = 1$, the reuse factor $\alpha(\pi, \psi)$ represents the number of pairs in ψ that use π. It is calculated as* $\alpha(\pi, \psi) := |\{(\pi, (\pi^\perp, \pi^\top)) \mid \pi = \pi^\perp \vee \pi = \pi^\top, (\pi^\perp, \pi^\top) \in \psi\}|$.

Relation to BDDs. A pair (tc, tc') of test cases showing the independence of some condition c_i has a vivid graphical interpretation on the BDD. It corresponds to a pair of paths $(\pi^\perp, \pi^\top)$ such that:

1. the tests evaluate the opposite truth values (i.e., $D(tc) = \neg D(tc')$);
2. $tc \leqq \pi^\perp, tc' \leqq \pi^\top$ (order wlog., the test cases may contain more input than strictly necessary).
3. both reach some node v_{c_i} using the same path through BDD(D) (i.e., $\pi^\perp[j] = \pi^\top[j]$ for $0 \leq j < i$);
4. their paths from v_{c_i} exit on either edge (i.e, $\pi^\perp[i] = \neg\pi^\top[i]$);
5. after v_{c_i}, both test cases take compatible choices along the paths for the remaining conditions, so that the independence property holds (i.e., $\pi^\perp[j] =_3 \pi^\top[j]$ for $i < j < n$).

This means especially that the two paths cannot cross (after the condition-node v_{c_i}), since this would immediately indicate an incompatible choice.

Figure 2 represents the overview on the selection of MC/DC pairs from the roBDD. The roBDD contains the root node labeled by R, non-terminal nodes labeled with conditions and two terminal nodes (0 and 1). The nodes are connected by solid and dashed edges representing assignments of 1 and 0 to each condition respectively. Every condition c may be represented multiple times on



(a) Pair selection (b) Reuse effect

Fig. 2: Overview on MC/DC pairs selection path from BDD and reuse effect

the BDD (nodes v_c). There may exist multiple paths to such a node. For every path reaching a (non-terminal) node, we attempt to extend it to construct pairs that show the independence effect of that condition. It is not guaranteed that the two complementary paths lead to opposite terminal nodes and our algorithm must explicitly check it step-by-step (modulo "don't care"-steps). The figure shows a representative of such pairs, $(\pi^{\perp}, \pi^{\top})$: they share the same prefix for all ordered conditions up to v_c. They then proceed in lock-step through the two branches to the terminals.

Figure 2 (b) illustrates some of the effects that we aim to achieve: as we search for pairs in the order of the roBDD, we will obtain some pair (shown in blue) from the heuristics (e.g. based on "longest path") which differs directly in the condition R for the root node. The next condition A in the order exists only in the left subtree, and we prefer a pair for it that reuses one of the previous path. Here, this can only be the left path for R, and hence we check if for the path that condition A shares with condition R we can construct a compatible path to the opposite terminal after leaving the node for A through the opposite edge (red pair). For condition B, we attempt to construct a pair by reusing the right branch for condition R (blue), and another one that uses the path that we used before both for R and A. We either take the only pair that fulfils our criteria, or again have the heuristics break a potential tie, here resulting in the green pair for condition B.

Due to the structure of roBDD, the derived test cases correspond to *MC/DC + short circuit* [6,5] where a test case can be composed with a three-valued assignment (*0: false, 1:true, and ?:not evaluated(a condition does not appear along the path)*). Therefore, to find the test cases that satisfy Unique Cause MC/DC [9], the "don't care" assignments will be replaced by either 0 or 1 pairwise (by the corresponding value at the same position in the partner path).

3.2 Algorithm and heuristics for test cases generation

Our approach for MC/DC test case generation for a decision D is based on the three-valued paths that are extracted from the equivalent roBDD. The MC/DC coverage criteria requires a pair of test cases that shows the independence effect

for every condition. The presence of "don't care" values in a BDD path gives us some flexibility when instantiating it to a test case and finding the complementary test case that leads to the opposite Boolean evaluation. As the wildcards may specialize to any Boolean value, we propose a greedy algorithm that tries to minimize the overall number of test case pairs for a decision D with n conditions from $2n$ to a value as close as possible to $n + 1$.

To this end, our method is divided in two stages: during the first phase, it initializes the MC/DC test suite with paths that are extracted from the BDD through any of our predefined heuristics, which intend to maximize the reuse factor in order to reduce the differences among test cases. Secondly, the selected BDD paths are specialized so that the wildcards take a concrete value while preserving the independence effect. We lift this property to sets of pairs of test cases with the definition of *instantiate* which computes the smallest set such that it guarantees that all members have been merged if possible:

$$
\begin{aligned}
&\forall (s, s') \in \mathit{instantiate}(S) : \\
&\quad \exists (p, p') \in S : uc3(p, p') \wedge s \leqq p \wedge s' \leqq p' && \text{(instantiated from } S\text{)} \\
&\quad \wedge\ \forall (p, p')\ \forall (q, q') \in S : s \leqq p \wedge p \leqq q \wedge p \leqq q' \Rightarrow s = p \wedge s' = p' && \text{(least upper bound)} \\
&\quad \wedge\ \exists c : \mathit{merge}_c(s, s') = s \wedge \mathit{merge}_c(s', s) = s' && \text{(fully merged).}
\end{aligned}
$$

This approach takes $n = |C|$ iterations, and each iteration adds a pair consisting of at most two new paths to the set. If S is empty, we can abort as this means there does not exist any pair showing the independence effect of that condition, and hence the decision D cannot be covered with the MC/DC-property. Correspondingly, unless we abort, the final set will contain n pairs, consisting of at most $2n$ individual paths. By construction, these pairs will provide three-valued MC/DC-coverage of the decision.

This leaves us two points to address: i) can we avoid constructing the set of *all* pairs for a condition, but instead only use a relevant, smaller subset as input to the heuristics, and ii) can we present evidence that our heuristics have a high likelihood of picking pairs that not only reuse a path from the already

Algorithm 1: MC/DC Test case generation

Input: An *roBDD* over conditions C with root r for a formula φ
Output: Set ψ of pairs of test cases that MC/DC-cover φ with $|C| + 1 \leq |\bigcup\{\{tc, tc'\} | (tc, tc') \in \psi\}| \leq 2|C|$.

1 $\psi = \emptyset$;
2 **forall** $c \in C$ **do**
3 Let $S := \{(\pi^{\top}_{v_c}, \pi^{\bot}_{v_c}) \mid$ where $\pi^{\top}_{v_c}, \pi^{\bot}_{v_c}$ are paths from the root r via some v_c to $\top$ and $\bot$ respectively, such that $[\pi^{\top}_{v_c}] \oplus_c [\pi^{\bot}_{v_c}]\}$.
4 Abort if $S = \emptyset$: no MC/DC cover of φ possible.
5 Let $(p, q) := \mathcal{H}(\psi, S)$ be the result of applying a given heuristics $\mathcal{H}$, such that $\exists (p', q') \in S : p = \mathit{merge}_c(p', q'), q = \mathit{merge}_c(q', p')$.
6 $\psi = \mathit{instantiate}(\psi \cup \{p, q\})$
7 **end**

selected pairs (if possible), but also contributes a fresh path that will be reused in the future. We address the first point through algorithmic construction, and evaluate the second through a series of experiments using the TCAS case study.

Algorithmic description. Any approach to a potentially optimal solution must reuse a test case that has already been selected as a partner in a pair for some other condition when selecting a pair for some other condition. It is hence clear that not all pairs for a condition may have to be constructed and evaluated. Rather, we first attempt to directly derive a pair from the existing set of test cases (by flipping only the corresponding condition), and only revert to deriving a new pair of completely fresh paths if such a derived path does not exist. Depending on the heuristics, identifying a completely fresh pair may entail a complete enumeration of pairs: it may be looking for the longest path with most reuse-potential (least number of "don't care"), which could ultimately be the last pair a given traversal of the BDD yields.

The representation as a BDD gives us an advantage in building fresh pairs: by exploring the tree from the root, the ordered labels tell us when we can preempt a search because the condition of interest does not exist in the remaining subtree, and we can continue our search in a sibling. Compared to an exploration of the corresponding truth-table, this effectively allows us to skip over irrelevant rows. We next formalize the notion of path-length in the roBDD.

Definition 14 (Length of a path/test case, $|\sigma|/|tc|$). *Given a path σ in the roBDD for a decision D from the root to a terminal, we denote the length of the path with $|\sigma|$. The length of a* test case *$|tc|$ is that of the underlying path.*

Note that since a test case can have more concrete inputs than are necessary for the path we have in the BDD, the length of a test case may be lower than the number of concrete inputs in that test case.

We propose five selection methods for test cases generation. All of them maximize the reuse factor ($\alpha()$) together with a second criteria, namely: the longest paths in BDD ($\mathcal{H}_{LPN}$, $\mathcal{H}_{LPB}$), the longest paths which may merge ($\mathcal{H}_{LMMN}$, $\mathcal{H}_{LMMB}$), and the longest paths with better size ($\mathcal{H}_{LPBS}$). Each type of heuristic implements two different flavors which sort the BDD paths depending on the interpretation of the reuse factor as a natural number ($\mathcal{H}_{LPN}$, $\mathcal{H}_{LMMN}$) or as a boolean value ($\mathcal{H}_{LPB}$, $\mathcal{H}_{LMMB}$)(e.g., $\alpha(p,\psi) < \alpha(q,\psi)$). We compare them with the *random reuser* ($\mathcal{H}_{RR}$) method as a baseline, which takes the first new path that forms a new pair with an existing test.

$\mathcal{H}_{LPN}/\mathcal{H}_{LMMN}$: This method chooses pairs of paths satisfying MC/DC based on the longest paths in BDDs with the highest reused factor. In case multiple pairs have equal reuse, we choose one where additionally the sum of the lengths is longest. The longest path or higher reuse factor may be better since it can be reused by many conditions that appear along the path.

$$\mathcal{H}_{LPN}(\psi, S) := (merge_c(p,q), merge_c(q,p)) \text{ where } (p,q) \in S$$

such that either (in order):

1. $\alpha(p,\psi) > 0 \wedge \alpha(q,\psi) > 0 \ \wedge \forall (p',q') \in S : \alpha(p',\psi) > 0 \wedge \alpha(q',\psi) > 0$
$\Rightarrow |p| + |q| \geq |p'| + |q'|$
(both test cases were already in the set)
2. $\forall (p',q') \in S : \alpha(p,\psi) + \alpha(q,\psi) \geq \alpha(p',\psi) + \alpha(q',\psi)$ (highest reuse)
$\wedge \ (\alpha(p,\psi) + \alpha(q,\psi) = \alpha(p',\psi) + \alpha(q',\psi) \Rightarrow |p| + |q| \geq |p'| + |q'|)$
(longest path).

$\mathcal{H}_{LPB}$/$\mathcal{H}_{LMMB}$: The previous heuristic $\mathcal{H}_{LPN}$ looks at the reuse of the paths in a pair: the existing path may have reuse > 0, and may occur in multiple pairs in the existing set. Its partner path may also be derived from another existing pair. Since it is not clear that past performance ("high reuse = used in multiple pairs by someone before") is an indication for future performance ("does it have more likelihood to be useful in future pairs?"), we also evaluate a variant that only prefers that there is *some* reuse, but not *how much*:

$\mathcal{H}_{LPB}(\psi, S) := (merge_c(p,q), merge_c(q,p))$ where $(p,q) \in S$:
$\alpha(p,\psi) + \alpha(q,\psi) > 0$ (has some reuse)
$\wedge \ \forall (p',q') \in S : \alpha(p',\psi) + \alpha(q',\psi) > 0 \Rightarrow |p| + |q| \geq |p'| + |q'|$(longest path).

The difference between this method and the previous one is that here we consider the reuse factor as Boolean. That is, we choose a pair with the longest paths in BDDs and we check if one of the paths is already reused as a part of an earlier pairs or not. This may give rise to greater non-determinism since more potential partners are considered equivalent.

Longest paths best size ($\mathcal{H}_{LPBS}$): selects MC/DC pairs where the paths have together the highest reuse and the sum of the lengths is strictly the longest.

4 Implementation of MC/DC test cases selection

In this section we describe how we evaluate our approach for the heuristics proposed in Section 3. For each heuristic, one run of Alg. 1 derives a set of test cases for a decision with MC/DC-coverage if it exists. Our heuristics are sensitive to exactly one parameter: the ordering of conditions when constructing the BDD. Furthermore, there is some inherent non-determinism: a heuristic picks randomly among equally best-ranked pairs. It is quite common to observe equivalent pairs with identical reuse and identical path-length. Secondary sources of non-determinism include e.g. iteration over unordered structures like sets which are implementation-specific to a given Python platform.

To give a proper evaluation, we control these in the following way: every heuristic is applied for a number of permutations of the order of the conditions for each decision. For decisions with a low number of conditions, we can hence even exhaustively evaluate the outcome of the heuristics for all permutations. In addition, we repeat a run on a given permutation, exploring different random choices within the equivalent best pairs.

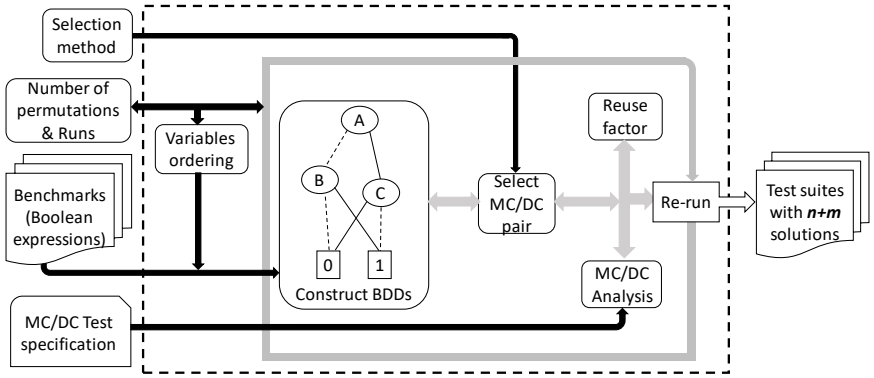


Fig. 3: Test cases generation framework

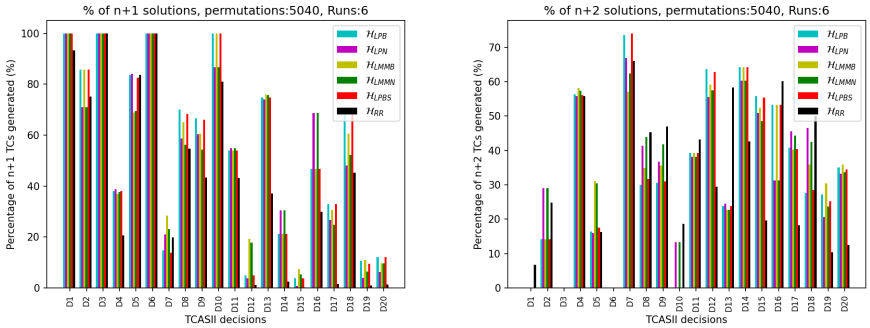
Our framework is based on the PyEDA library [13] and implemented in Python. We test our algorithm on the Traffic Alert and Collision Avoidance System (TCAS II) benchmark [33,25] which has been frequently used in literature [19,21,37,22,17]. The benchmark refers to specifications written as Boolean expressions (decisions) which are logically evaluated to true or false depending on the truth values assigned to the contained conditions.

Below, we present detailed results for a well-known set of TCAS II decisions that can be reproduced with the code in our open source repository [3]. We do not report execution times for our experiment, as our implementation is not optimized in any way beyond obvious algorithmic constructions to minimize BDD-traversal.

4.1 Experimental setup

Fig. 3 shows our test cases generation framework. Our setup takes as input the roBDD for a given decision, the number of permutations, and the number of runs that we perform for each process of test cases generation. The selection method refers to the different heuristics proposed in Section 3: $\mathcal{H}_{LPN}$, $\mathcal{H}_{LPB}$, $\mathcal{H}_{LMMN}$, $\mathcal{H}_{LMMB}$, $\mathcal{H}_{LPBS}$ and $\mathcal{H}_{RR}$. The benchmarks refer to the specifications written as Boolean expressions (decisions) which are logically evaluated to true or false depending on the truth values assigned to the contained conditions. MC/DC test specifications are the meaning of what is MC/DC in the context of roBDDs and three values logic (cfr. Theorem 1 and Def. 9). We consider the reuse factor in our MC/DC analysis to reuse as much as possible the existing selected TCs and finally, we produce n MC/DC pairs as output for each decision with the size of n+m solutions. Our results show that we produce mostly $n+1$ solutions and the rest of solutions are less than $2n$ with 100% MC/DC.

[3] https://github.com/selabhvl/py-mcdc/



(a) % of n+1 solutions (b) % of n+2 solutions

Fig. 4: Comparison of % for n+1 and n+2 solutions for different heuristics

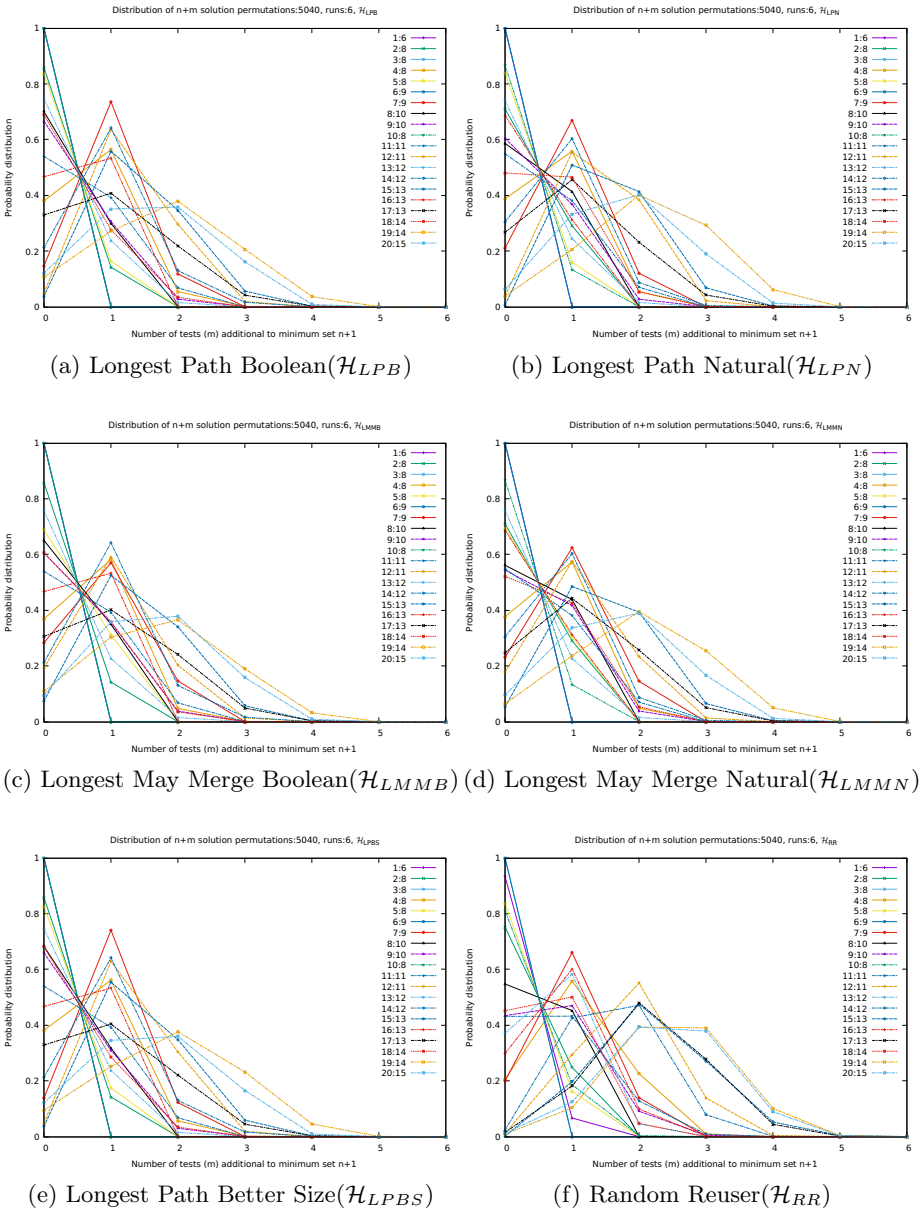
4.2 Experimental Results

Figure 4(a) and (b) present our results as the percentage of generated solutions of sizes $n+1$ and $n+2$ for TCAS II based on our heuristics and the baseline RR heuristic. We consider 5040 different orders at most for each decisions (this exhaustively covers all orders for decisions with up to seven conditions). This sample size already yields evidence that repeated application of the algorithm to different orders will discover a (close to) optimal solution reasonably quickly.

For each heuristic we collect all possible sets of MC/DC covering test cases. MC/DC coverage is calculated as the percentage of the number of covered conditions to the total number of conditions in a decision. In case the MC/DC coverage percentage is less than 100%, it means that MC/DC is not fulfilled for that decision. We present results for solutions of size $n+1$ (optimal) and $n+2$ for our heuristics as shown in Figure 4. The charts for the heuristics can be reproduced from our open repository. From the TCAS II benchmark results in Figure 4 and 5, we highlight the following:

1. Our heuristics find the test suite sets of $n+1$ solutions for each decision, whereas $\mathcal{H}_{RR}$ failed to find any minimal solution for D15. Our heuristics perform better compared to $\mathcal{H}_{RR}$ for 18 out of 20 decisions and have equal results for two decisions in terms of which heuristic has frequently the highest of $n+1$ solutions with 100% MC/DC. This shows that the approach of permuting order is a viable strategy to eventually obtain an optimal results.
2. $\mathcal{H}_{LPB}$ and $\mathcal{H}_{LMMB}$ out-perform all others with 10 cases (50%) having the highest % of n+1 solutions.
3. Comparing the $\mathcal{H}_{LPB}$ to $\mathcal{H}_{LMMB}$,$\mathcal{H}_{LPB}$ is 2 cases (10%) higher than $\mathcal{H}_{LMMB}$.
4. We observed that $\mathcal{H}_{LMMN}$ is 2 cases (10%) higher than $\mathcal{H}_{LPN}$.
5. $\mathcal{H}_{LPBS}$ has better results in some decisions than $\mathcal{H}_{LMMN}$ and $\mathcal{H}_{LPN}$.
6. In three decisions (D2, D5 and D7), $\mathcal{H}_{RR}$ has better results than some of our our heuristics. We attribute this outcome to random chance.

7. From Figure 4 (b) which represent the n+2 solutions, we can see that for the decisions in which we did not find the highest percentage of n+1 solutions now we have a high % of n+2 solutions, which indicates that our test suites generated are closer to lower bound (n+1) of MC/DC minimal set.



(a) Longest Path Boolean($\mathcal{H}_{LPB}$)

(b) Longest Path Natural($\mathcal{H}_{LPN}$)

(c) Longest May Merge Boolean($\mathcal{H}_{LMMB}$)

(d) Longest May Merge Natural($\mathcal{H}_{LMMN}$)

(e) Longest Path Better Size($\mathcal{H}_{LPBS}$)

(f) Random Reuser($\mathcal{H}_{RR}$)

Fig. 5: Probability distribution of n+m solutions for 5040 permutations, 6 runs

In summary, our results show that we produce mostly $n+1$ solutions and the rest of solutions are less than $2n$ with 100% MC/DC adequacy. Figures 5 (a)-(f) show the probability distribution of $n+m$ TCs generated for 5040 permutations, 6 runs for different heuristics. The x-axis shows the number of test cases (m) *additional* to the minimal solution ($n+1$). The labels show the decision number and the contained conditions as presented in [25]. All the solutions are have less than $2n$ test cases, as the maximum observed for m is 6 while the range of number of conditions is 6 to 14. Figures 5 shows that most solutions are much closer to the minimal size (to the left) than to the worst case.

Another challenge which is not directly related to our approach but to MC/DC is the coupled and masked conditions where it is difficult to get a full MC/DC coverage with masked condition. For example the decision D10 in the TCAS II benchmark has two conditions (b and h) which are masked. Out of the nine conditions, hence only seven are retained in the roBDD and we compute our minimal solution accordingly.

For the complex example D15 in the 20 TCAS II decisions, our algorithm takes on average 0.7 seconds (incl. time for constructing the BDD) for a single run on an Intel(R) Core(TM) i7-7700 CPU @3.60GHz Linux machine with 64 GB RAM. From the proposed heuristics, we recommend the longest paths with reuse as Boolean number($\mathcal{H}_{LPB}$) as it shows high performance both in terms of high percentage of $n+1$ solutions and short time to compute the solutions compared to the rest of the heuristics.

5 Related work

Automatic test data generation approaches were proposed in [16,4,36] and are based on greedy or meta-heuristic search strategy. They use search algorithms to extract test paths from the control flow graph of a program, then invoke an SMT solver to generate test data [16] and afterwards reduce the test-suite with a greedy algorithm. The drawback for this approach is that often infeasible paths are selected, resulting in significant wasted computational effort. We did not investigate test *data* generation here, only boolean inputs to a single decision.

Kitamura et al. [25] and Yang et al. [37] use a SAT solver to construct minimal MC/DC test suites. That is, the MC/DC criterion is encoded in a single query, and the solver produces a suitable assignment for test case inputs if it exists, or times out. In contrast to the exhaustive nature of SAT queries which may lead to timeouts, our approach delivers a single answer in much less time, but may require repetition to find an optimal solution. Some of their results do not satisfy UC-MC/DC in some cases, and generate test cases only for Masking MC/DC. There are also some conditions which are reported as infeasible, while the MC/DC pairs for those conditions can be found. For example in [25], decisions 6 and 8 of the TCAS II benchmarks have test suites with 3 and 4 test cases for 8 and 9 conditions respectively which cannot satisfy MC/DC.

A study of enhanced MC/DC coverage criterion for software testing based on n-cube graphs and gray code is presented in [8]. It is an exhaustive approach that takes input as a Boolean expression, builds the n-cube graph, and deduces

test cases from all vertices of the graph. Their test cases selection is based on the weight of each test case in a similar way that we calculate the reuse factor of a path. The main difference is that they have to construct the n-cube graph which have the same effect as exhaustive traversal of a truth table and the resulting size of the test suite is not minimal.

Gay et al. [14,15], developed a technique to automatically generate test cases using model checkers for masking MC/DC. Using the JKind model checker, they produce a list of all test inputs and then select the desired test cases while preserving the coverage effectiveness. Their test suite reduction algorithm used to reduced the original test-suite does not guarantee to find the smallest set. They tested their approaches on different real-world avionics systems where they achieved an average MC/DC coverage of 67.67%.

Comar et al. [12] discussed MC/DC coverage in terms of BDD coverage. They examine the set of distinct paths through the BDD that have been taken based on the control flow graph. Based on BDDs they investigated the formalization and comparison of MC/DC to object branch coverage, but the test cases selection is out of their scope. We extend the formalization and definitions of MC/DC in terms of BDDs in the context of test cases selection.

The roBDDs have been used in [22,17] for test cases generation, and highlight the properties and benefits of roBDDs, however, MC/DC was not considered as coverage criterion. Like our approach, their greedy approach incrementally selects a pair of paths where only one condition changes for every condition.

6 Conclusion and Future works

We presented a heuristics-based approach for generating test cases for a Boolean decision (given as roBDD) that satisfy the MC/DC criterion. We evaluate our approach on the TCAS II Benchmark and results shows that we frequently find solutions which are equal or close to the minimal number of test cases without expensive back-tracking.

Our approach is sensitive to variable ordering in the BDD as each order yields a different roBDD. We obtained MC/DC solutions of size $n + 1$ by performing few permutations of conditions in a decision for all tested decisions. We present also the other possible solutions which show full MC/DC coverage. In general, our solutions have a size ranging from $n + 1$ to $2n$, with a high percentage of size $n + 1$ or $n + 2$ solutions, where even the latter, although not optimal, may be acceptable to a user. We proposed different heuristics and compared their properties. All our heuristics perform better than $\mathcal{H}_{RR}$. $\mathcal{H}_{LPB}$ and $\mathcal{H}_{LMMB}$ out-perform all other heuristics with 10 times (50%) having highest percentage of $n + 1$ solutions. We recommend $\mathcal{H}_{LPB}$ since it is 10% better than $\mathcal{H}_{LMMB}$.

For the future work we plan to extend our algorithm so that we support data input coverage where conditions are not abstracted, which requires taking constraints into consideration. We will also attempt to integrate our test case generation algorithm into our MC/DC measurement tool and model[2,3]. Although the experimental data shows that we always find an optimal solution, it remains open if this is a general property of our approach.

References

1. Adacore. Technical report on OBC/MCDC properties. Technical report, Couverture project, 2010.
2. Faustin Ahishakiye, Svetlana Jakšić, Volker Stolz, Felix D. Lange, Malte Schmitz, and Daniel Thoma. Non-intrusive MC/DC measurement based on traces. In *International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 86–92. IEEE, 2019.
3. Faustin Ahishakiye, José I. Requeno Jarabo, Lars Michael Kristensen, and Volker Stolz. Coverage analysis of net inscriptions in coloured Petri net models. In *International Conference on Verification and Evaluation of Computer and Communication Systems (VECoS)*, pages 68–83. Springer, 2020.
4. Zeina Awedikian, Kamel Ayari, and Giuliano Antoniol. MC/DC automatic test input data generation. In *Annual Conference on Genetic and Evolutionary Computation Conference (GECCO)*, pages 1657–1664. ACM, 2009.
5. Matteo Bordin, Cyrille Comar, T. Gingold, Jérôme Guitton, Olivier Hainque, and Thomas Quinot. Object and source coverage for critical applications with the COUVERTURE open analysis framework. In *European Congress Embedded Real Time Software and Systems (ERTS)*, pages 1–9, 2010.
6. Certification Authorities Software Team (CAST). Rationale for accepting masking MC/DC in certification projects. *Technical Report: Position Paper CAST-6*, 2001.
7. Certification Authorities Software Team (CAST). What is a "Decision" in application of Modified Condition/Decision Coverage (MC/DC) and Decision Coverage (DC)? *Technical Report: Position Paper CAST-10*, 2002.
8. Jun-Ru Chang and Chin-Yu Huang. A study of enhanced MC/DC coverage criterion for software testing. In *Annual International Computer Software and Applications Conference (COMPSAC)*, pages 457–464, 2007.
9. John J. Chilenski. An investigation of three forms of the Modified Condition Decision Coverage (MC/DC) criterion. Technical report, Office of Aviation Research, 2001.
10. John J. Chilenski and Steven P. Miller. Applicability of Modified Condition/Decision Coverage to software testing. *Software Engineering Journal*, 9(5):193–200, 1994.
11. Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.
12. Cyrille Comar, Jérôme Guitton, Olivier Hainque, and Thomas Quinot. Formalization and comparison of MC/DC and object branch coverage criteria. In *European Congress Embedded Real Time Software and Systems (ERTS)*, pages 1–10, 2011.
13. Christopher R. Drake. PyEDA: Data structures and algorithms for electronic design automation. In *Python in Science Conference (SciPy)*, 2015.
14. Gregory Gay, Ajitha Rajan, Matt Staats, Michael Whalen, and Mats P. E. Heimdahl. The effect of program and model structure on the effectiveness of MC/DC test adequacy coverage. *ACM Transactions on Software Engineering and Methodology*, 25(3), July 2016.
15. Gregory Gay, Matt Staats, Michael Whalen, and Mats P. E. Heimdahl. The risks of coverage-directed test case generation. *IEEE Transactions on Software Engineering*, 41(8):803–819, 2015.
16. Kamran Ghani and John A. Clark. Automatic test data generation for multiple condition and MC/DC coverage. In *International Conference on Software Engineering Advances (ICSEA)*, pages 152–157, 2009.

17. Hongfang Gong, Junyi Li, and Renfa Li. CTFTP: A test case generation strategy for general Boolean expressions based on ordered binary label-driven Petri nets. *IEEE Access*, 8:174516–174529, 2020.
18. Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry. Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack. *Empirical Software Engineering*, 24(2):674–717, 2019.
19. Sylvain Hallé, Edmond La Chance, and Sébastien Gaboury. Graph methods for generating test cases with universal and existential constraints. In *International Conference on Testing Software and Systems (ICTSS)*, pages 55–70. Springer, 2015.
20. Alan J. Hu. Formal hardware verification with BDDs: an introduction. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, volume 2, pages 677–682. IEEE, 1997.
21. James A. Jones and Mary J. Harrold. Test-suite reduction and prioritization for Modified Condition/Decision Coverage. *IEEE Transactions on Software Engineering*, 29(3):195–209, 2003.
22. Akram Kalaee and Vahid Rafe. An optimal solution for test case generation using ROBDD graph and PSO algorithm. *Quality and Reliability Engineering International*, 32(7):2263–2279, 2016.
23. Susanne Kandl and Sandeep Chandrashekar. Reasonability of MC/DC for safety-relevant software implemented in programming languages with short-circuit evaluation. *Computing*, 97(30):261–279, Mar 2015.
24. Sekou Kangoye, Alexis Todoskoff, and Mihaela Barreau. Practical methods for automatic MC/DC test case generation of Boolean expressions. In *IEEE AUTOTESTCON*, pages 203–212. IEEE, 2015.
25. Takashi Kitamura, Quentin Maissonneuve, Eun-Hye Choi, Cyrille Artho, and Angelo Gargantini. Optimal test suite generation for Modified Condition Decision Coverage using SAT solving. In *Computer Safety, Reliability, and Security*, pages 123–138. Springer, 2018.
26. Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 1998.
27. Jim Newton and Didier Verna. A theoretical and numerical analysis of the worst-case size of reduced ordered binary decision diagrams. *ACM Transactions on Computational Logic*, 20(1), January 2019.
28. Frederic Pothon. DO-178C/ED-12C versus DO-178B/ED-12B: Changes and Improvements. Technical report, AdaCore, 2012. available at https://www.adacore.com/books/do-178c-vs-do-178b.
29. Sherief Reda and Ashraf M. Salem. Combinational equivalence checking using Boolean satisfiability and binary decision diagrams. In *Design, Automation and Test in Europe. Conference and Exhibition (DATE)*, pages 122–126. IEEE, 2001.
30. Leanna Rierson. *Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance*. CRC Press, 2013.
31. Gregory Tassey. The economic impacts of inadequate infrastructure for software testing, 2002.
32. Sergiy Vilkomir and Jonathan Bowen. Reinforced Condition/Decision Coverage (RC/DC): A new criterion for software testing. In *International Conference of B and Z Users*, volume 2272 of *LNCS*, pages 291–308. Springer, 2002.
33. Elaine Weyuker, Tarak Goradia, and Ashutosh Singh. Automatically generating test data from a Boolean specification. *IEEE Transactions on Software Engineering*, 20(5):353–363, 1994.

34. Elaine J. Weyuker, Stewart N. Weiss, and Dick Hamlet. Comparison of program testing strategies. In *Symposium on Testing, Analysis, and Verification (TAV)*, pages 1–10. ACM, 1991.
35. James Worrell. Logic and Proofs-Binary Decision Diagrams. Available at https://www.cs.ox.ac.uk/people/james.worrell/lec5-2015.pdf.
36. Tianyong Wu, Jun Yan, and Jian Zhang. Automatic test data generation for unit testing to achieve MC/DC criterion. In *International Conference on Software Security and Reliability (SERE)*, pages 118–126. IEEE Computer Society, 2014.
37. Ling Yang, Jun Yan, and Jian Zhang. Generating minimal test set satisfying MC/DC criterion via SAT based approach. In *Annual ACM Symposium on Applied Computing (SAC)*, pages 1899–1906. ACM, 2018.