Test Case Generation for Boolean Expressions by Cell Covering

Lian Yu, *Member, IEEE,* and Wei-Tek Tsai, *Member, IEEE,*

Abstract—This paper characterizes Boolean expression faults as changes of the topological structures in terms of shrinking and/or expanding regions in K-map. A *cell-covering* is a set of cells (test cases) in K-map to cover the fault regions such that faults guarantee to be detected. Minimizing cell covering can be formulated as an Integer Linear Programming (ILP) problem. By analyzing the structures of the constraint coefficient matrix, the original problem can be decomposed into sub-programs that can be solved instead of the original problem, and this significantly reduces the time needed for ILP execution. An efficient approximate algorithm with a tight theoretical bound is used to address those complex Boolean expressions by corresponding the cell-covering problem to the set-covering problem. The optimal approach and the approximate approach are combined into a hybrid process to identify test cases based on the fraction analysis on the ILP relaxation. The proposed approach is evaluated by three sets of Boolean expressions and the results are compared with three leading approaches with respect to test sizes, time consumption and fault detection capabilities. For most Boolean expressions encountered, the proposed approach obtains optimal solutions quickly, and produces near-optimal solutions rapidly for those rare and complex expressions.

Index Terms—Boolean expression testing, fault characterization, cell-covering problem, approximate algorithms

1 Introduction

1.1 Background and Motivations

Boolean expressions are widely used in software including requirements, design, and code to express logic conditions in applications. For example, Boolean expressions often appear in business rules in requirement specifications, the guard conditions in state machines, and the *if*, *while* statements of Java/C++ languages, by mapping each of relational expressions if any, like $i > 0$, to a Boolean variable.

The correctness and validity of Boolean expressions lay the foundation of the correctness and robustness of those software applications. Given a Boolean expression with n variables, exhaustive testing requires 2^n distinct test cases, and the test size will grow in an exponential order with the increasing number of the variables. Selecting a small and effective subset from all the possible test cases has been an important research topic [3], [4], [6], [17], [34], [37], [39], [43].

Example 1. Let a represent for valve A, x for cabin temperature, c for valve C, and y for wind-force. A requirement might say that "if either a is on and x is greater than T units, or c is on and y is greater than or equal to W units, cruise parameter S_1 should be adjusted, otherwise keep unchanged." By mapping "$x > T$" to b, and "$y \geq W$" to d, whether S_1 should be adjusted or not can be represented logically as a Boolean expression, $S_1 = ab + cd$, where ab refers to "a and b", cd refers to "c and d", and $ab + cd$ refers to "ab or cd". However, the expression can be written incorrectly in different stages, e.g., specification, design or implementation, and in different ways. A fault may occur in S_1 due to: incorrectly understanding/writing "*or*" as "*and*"; or omitting term cd; or inserting literal a into term cd; unconsciously negating S_1; or negating term ab; or negating literal b, even replacing literal b with literal c; or by chance, writing term ab as $a + b$; or omitting literal a.

To deal with such Boolean expression faults, researchers have worked on three aspects: (1) classify the types of the faults according to their attributes; (2) create a hierarchy of these types of faults according to their subsuming relations; (3) design test strategies to detect these faults.

In fact, the nine faults in Example 1 fail into nine classes/types of Boolean expression faults, i.e., "+" operator reference fault (ORF+), term omission fault (TOF), literal insertion fault (LIF), expression negation fault (ENF), term negation fault (TNF), literal negation fault (LNF), literal reference fault (LRF), "·" operator reference fault (ORF·), and Literal omission fault (LOF) [10], [25], [32], [34], [36], [38].

The subsuming relations among these single fault classes reveal the facts that test cases of a test technique that can detect the faults on the up-level of the class hierarchy can also detect the faults in the low-level. It can be simply understood that the test case set that guarantee to detect the faults in up-level is a subset of that of low-level under certain conditions if any. Thus fault class hierarchy provides a useful theoretical framework to evaluate the effectiveness of fault-based testing techniques.

- *L. Yu is with the School of Software and Microelectronics in Peking University, Beijing 102600, China. E-mail: lianyu@ss.pku.edu.cn.*
- *W.-T. Tsai is with Beihang University, Beijing 102600, China, and was with School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ 85281. E-mail: wtsai@asu.edu.*

Manuscript received 29 Mar. 2015; revised 5 Feb. 2017; accepted 7 Feb. 2017. Date of publication 13 Feb. 2017; date of current version 5 Jan. 2018. Recommended for acceptance by A. Zeller.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2017.2669184

In particular, Kuhn [25] identified subsuming relations among three types of faults. The ordering in the hierarchy is consistent with the ordering of the effectiveness of various fault-based testing techniques for detecting these faults as found in previous empirical studies by Vouk et al. [42] and Weyuker et al. [43]. Tsuchiya and Kikuno [38] added one more fault type in the hierarchy later. Lau and Yu [26] further analyzed the relationships between variable and literal faults, and among literal, operator, term, and expression faults, and extended and complemented the hierarchy [25], [38] including nine types of faults. Among them, LIF, LOF and LRF sitting on the top of the fault hierarchy subsume other types of faults, i.e., if a test criterion guarantees detecting these three types of faults, it can guarantee to detect all types of faults in the hierarchy.

Researchers have also developed several test strategies and algorithms to generate test suites to detect the faults. Regarding the single faults in the hierarchy, MC/DC (Modified Condition/Decision Coverage) [6] tests guarantee to detect ENFs and TNF, but do not guarantee to detect most single faults in Lau and Yu's hierarchy [17]. Weyuker et al. [43] proposed the MAX-A and MAX-B test criteria that guarantee to detect all single faults in the hierarchy.

Chen et al. [4] developed the MUMCUT test criterion consisting of three constituent criteria: multiple unique true point (MUTP), multiple near false point (MNFP), and corresponding unique true point and near false point (CUTPNFP). Compared with MAX-A and MAX-B test criteria, Chen et al. provided theoretically the proofs that MUMCUT guarantees the detection of LIF, LOF and LRF faults, and by implementing MUMCUT coverage criterion using a heuristic approach, they showed that their empirical results based on MUMCUT produced less test cases while guaranteeing the detections of all faults in the fault hierarchy.

Kaminski and Ammann [18], [20] explored the relationships between LIF and LRF, as well as LRF and LOF, developed minimal-MUMCUT, and implemented the criterion with a heuristic approach that produced even less test cases than MUMCUT while guaranteeing the detections of all faults in the fault hierarchy. In [19], Kaminski and Ammann formulated minimal-MUMCUT strategy as 0-1 integer programming with the constraints to satisfy the three constituent criteria, MUTP, MNFP and CUTPNFP by taking into consideration the feasibility of MUTP and CUTPNFP; and the objective function to minimize the sum of test cases meeting such constraints.

Being able to obtain the minimum test sizes while maintaining fault detection capabilities is ideal and effective. However, the optimization model based on MUMCUT, as [19] does, has the following issues:

(1) MUMCUT takes test cases (or test points) only from unique true points (UTPs) and near false points (NFPs) [26]. Eight types of double faults [27], [28], [29] (two faults, either the same or different types, occur simultaneously) in the fault hierarchy cannot be detected by MUMCUT [20], as well as the optimal minimal-MUMCUT in [19]. For example, one of such double faults is due to two LIFs occurring in two terms of an expression at once, and can be detected by test cases only from overlapping true points (OTPs); but both MUMCUT and optimal minimal-MUMCUT never take test cases from OTPs. Therefore, a different fault detection technique needs to be developed that can detect both the single faults and double faults in a *general* way while achieving optimality regarding test sizes.

(2) Optimization models of fault detections need to be investigated regarding computation time. How to leverage the characteristics of optimization model structures by decomposing problems into a set of independent sub-problems such that reduced sub-problems can be quickly solved is an open question.

(3) When a problem is large and cannot be decomposed as described in (2), a hybrid approach is needed to solve the problem in an efficient way. The challenge with the hybrid approach is to decide when to choose an optimal approach and when to choose an approximate approach.

The above issues motivate the research in this paper. The goal is to provide a fault modelling approach that must reflect the *real* or *valid* changes of expressions, i.e., it can reveal the equivalent mutations of expressions, systematically capturing these *valid* changes/faults, formulizing these changes as mathematical constraints, and finally solving the problems with the objective of minimizing the test sizes. The merit of capturing the faults as the changes of spaces is that the fault detections are not dependent on any specific test strategies, such as MC/DC, MAX-A&MAX-B, MUMCUT, or minimal-MUMCUT, as it targets faults directly.

1.2 Contributions

This paper contributes in the following ten aspects:

(1) *Fault characterization by topological structure in K-map:* These faults of Boolean expressions can be characterized by their changes in topological structures, i.e., shrinking or/and expanding regions with respect to the expressions in a K-map, see Section 4. This is the first time Boolean faults can be viewed topologically: e.g., LIFs will incur shrinking regions; LOFs will incur expanding regions; and LRFs will incur both shrinking regions and expanding regions.

(2) *Fault detections by examining topological structure:* An important consequence of (1) is that, one can generate test cases based on topological structure that will surely detect these single and double faults of the nine types. The *differentials* between the correct K-map and the faulty one regarding the topological structure are their corresponding test cases. This is non-trivial as one cell may cover multiple faults in a multi-dimensional K-map, and it is necessary to distinguish the role of each cell with respect to different faults and topological structure.

(3) *Optimal solution to detect these faults:* Based on (1) and (2), one can obtain global optimal solutions, rather than local optimal solutions, by solving a cell-covering problem formulated as an Integer Linear Programming (ILP), see Section 6.1.

(4) *Automatically generating the constraint coefficient matrix for ILP:* This paper designs algorithms to automatically generate the constraint coefficient matrices of

ILPs to detect single and double faults of LIFs, LOFs and LRFs based on their topological structures, see Section 6.2 and Appendix C, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2017.2669184. This is the first time that the ILP formulation can be automatically generated for cell-covering problems, and the algorithms are involved as one cell may cover multiple faults in multi-dimensional K-maps.

(5) *Formal proofs of relationships of cell-covering for LIFs & LRFs, and LRFs & LOFs:*
   a) *Relationship between LIFs & LRFs:* This paper proves that if a valid shrinking region of LIF is not empty, cells in the region guarantee to detect LRF as described in Section 7.1 and **Theorem 4** in Appendix B5, available in the online supplemental material. Thus, one can significantly reduce both the number of rows of coefficient matrix and the number of variables in ILP, saving much time to generate the constraints and obtain optimal solutions.
   b) *Relationship between LRFs & LOFs:* If the valid corresponding expanding region of LRFs must be needed and is not empty, cells in the region guarantee to detect LOF as described in Section 7.1 and **Theorem 5** in Appendix B6, available in the online supplemental material. This property can be leveraged to reduce significant time needed to generate the constraints of ILP.
   c) *Relationship between double faults:* It is interesting to notice that a single fault of a valid shrinking of term may be *re-filled* by a valid expanding of another term, as a result, such a double fault does not make any changes of the topological structure of an expression. On the other hand, a double fault may incur a fault that cannot be detected by neither MUMCUT, minima-MUMCU, nor the optimal minimal-MUMCUT [19], [20]. The detections of double faults are described in Section 7.2.

(6) *Optimization by diagonalizing the constraint coefficient matrix:* By linearly transforming the ILP matrix into a diagonal matrix, each block on the diagonal line corresponds to an independent sub-problem of the original ILP. In this way, the original problem can be solved by solving sub-problems first, and putting optimal solutions of the sub-problems together to obtain the optimal solution for the original problem. These significantly reduce the time needed to obtain optical solutions, see Section 7.3.

(7) *Approximate approach:* For complex cell-covering sub-problems, approximate approaches are applied by converting into the set-covering algorithm as described in Section 8.4. Different from heuristics that usually find reasonably good solutions efficiently, approximate approaches can find provable quality solutions with provable run-time bounds [45], and thus one can have much better confidence. This paper utilizes a greedy algorithm as an approximate solution with a known bound, and solutions produced are near optimal.

(8) *Hybrid approach:* An integrated hybrid process is developed to balance the solution optimality and the time consumptions as described in Section 9. Whether to use the optimal approach or to use the approximate algorithm depends on the difficulties of sub-problems defined in Section 7.4. The difficulty assessment indicates that the size alone is not the best indicator, instead it is the structure. It is interesting to notice that by diagonalization of the constraint coefficient matrix, the approximate algorithm is not applied to the whole cell-covering problem, instead to the non-trivial sub-problems with larger sizes but smaller than the original problem, while the trivial sub-problems can obtain the optimal solutions, see Sections 8.3 and 8.4.

(9) *Tool development:* A tool [51] has been implemented to automate the hybrid process, including generation of constraint coefficient matrices, diagonalization and optimization, implementation of approximate algorithms and integration of GLPK [48] ILP solver.

(10) *Empirical experimentation and evaluation:* This paper analyzes three data sets, and compares fault detection capabilities and the performance with three leading approaches, see Section 10. The data size is the largest when compared with the ones used in the literature.

1.3 Paper Organization

Section 2 presents the related work; Section 3 presents definitions and notations; Section 4 characterizes Boolean expression faults with respect to the changes in topology structure; Section 5 describes the conditions to detect LIFs, LOFs, and LRFs that are on the top of the fault hierarchy; Section 6 describes the cell-covering problem and ILP formulation; Section 7 describes the optimizations to solve the ILP; Section 8 depicts approximate approaches for complex cell-covering problems; Section 9 illustrates a hybrid approach; Section 10 demonstrates the experiments with three data sets; Section 11 discusses the threats of validity; and Section 12 concludes this paper. Appendix A, available in the online supplemental material, lists the mathematical notations and abbreviations that are used in the main text and appendixes; Appendix *B*, available in the online supplemental material, formally defines and proves the fault detection conditions; Appendix C, available in the online supplemental material, formally presents the algorithms to automatically generate the constraint coefficient matrices to detect the target faults; and Appendix D, available in the online supplemental material, presents the relationship between TRF-TIF and a greedy algorithm.

2 Related Work

Fault-based testing of logical expressions [10], [35], particularly Boolean expressions [4], [17], [33], [43] have received significant attention. In contrast to other testing methods, the fault-based testing hypothesizes common types of faults that may be committed by programmers, and designs test cases to address these faults [30].

Various types of faults and their relations have been studied [10], [11], [25], [32], [34], [36], [38], [43], [46]. According to Kuhn [25], if a specification is expressed in a Boolean

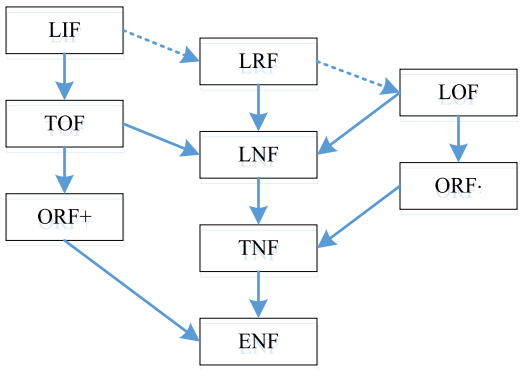


Fig. 1. Fault class hierarchy of Boolean expressions.

expression S in disjunctive normal form (DNF, e.g., $S = ab + cd + abc$), a test case that can detect a variable-reference fault (VRF: a Boolean variable replaced by another Boolean variable, e.g., ab changed to ac) can also detect a variable-negation fault (VNF: a Boolean variable replaced by its negation, e.g., ab changed to $a\bar{b}$), that in turn can detect the corresponding ENF (a Boolean expression replaced by its negation, e.g., S changed to $\bar{S}$). He built a fault hierarchy and the ordering in the hierarchy reflects the ordering of the effectiveness of various fault-based testing techniques, and the ordering is consistent with the studies by Vouk [42] and Weyuker [43].

Tsuchiya and Kikuno [38] analyzed the relationship between a VRF and a missing-condition fault (MCF: a condition omitted from the implementation of specification, e.g., ab changed to a). They discovered that test cases that can detect a MCF may not detect the corresponding VRF, and vice versa. Kuhn [25], Tsuchiya and Kikuno [38] clarified the relations among MCFs, VRFs, VNFs, and ENFs.

Many other faults, such as those involving logical operators [2], [34], [36], [43], terms and literals [3], [5], have not been considered [25], [38]. Lau and Yu [26] extended and complemented the work in [25], [38] by analyzing the relationships between variable and literal faults, and among literal, operator, term, and expression faults. They built an extended fault class hierarchy in which a test case that detects a fault f will always detect the corresponding fault (s) at the child positions (in terms of arrow directions) of fault f in the hierarchy.

Fig. 1 displays a modified version of Lau and Yu's fault hierarchy [26], based on whether an LRF (e.g., ab changed to ac) can be detected by an NFP or a UTP [20]. LRF is called as VRF in [24], [43]. It is noticed that in some studies, e.g., in [25], VNF and VRF means the fault that replaces *every* occurrence of the variable in the expression by its negation and another variable, respectively.

The hierarchy includes nine types of faults: ENF, TNF (e.g., ab changed to $\overline{ab}$), LNF (e.g., ab changed to $\bar{a}b$), LRF, ORF· (e.g., ab changed to $a + b$), LOF (e.g., ab changed to a), ORF+ (e.g., $ab + cd$ changed to $abcd$), TOF (e.g., $ab + cd$ changed to cd), and LIF (e.g., ab changed to abc) [26].

A solid arrow from a source fault (e.g., LRF) to a destination fault (e.g., LNF) indicates that if a test can detect the source fault, it also can detect the corresponding destination fault. These faults (e.g., LIF, LRF, and LOF) represent large classes of faults as they dominate many other faults (e.g., TOF, LNF, ORF·). When a term of a Boolean expression in DNF is a *min-term*, i.e., each variable has an appearance in the term, there is no LIF and LRF for that term. In this case, a test case to detect TOF must be generated to detect faults in the hierarchy.

Yu and Lau [41] compared MC/DC [6], MUMCUT and other coverage criteria for logical decisions by formal and empirical analysis, focusing on the fault-detecting ability of test sets. They pointed out that if a logical decision S is in IDNF (Irredundant DNF, e.g., $S = ab + cd$), then a test set satisfying any one of DC (Decision Coverage), C/DC (Condition Decision Coverage) or MC/DC will always detect ENF and TNF. CC (Condition Coverage) does not guarantee to detect ENF and TNF. Their results show that MC/DC is efficient, but its test sets may still miss faults that can almost always be detected by test sets satisfying MUMCUT [41].

When an LRF can be detected by an NFP only, a test set detecting all LIFs does not guarantee to detect all LRFs because one or more equivalent LIFs may exist for some of those LRFs. Thus, the solid arrow between the LIF and LRF in Lau and Yu's hierarchy is changed to a dashed arrow in [20]. In addition, when an LRF can be detected by an NFP only, adding a test to detect that LRF guarantees detection of the corresponding LOF [20] as indicated by a dashed arrow between the LRF and LOF [41].

With the hierarchy among fault classes in DNF, Kaminski and Ammann exploited the relations as indicated with dashed arrows between LIF and LRF and between LRF and LOF in the minimal-MUMCUT strategy to obtain test suites with the same fault detection capability while having fewer test cases than MUMCUT [4], [18]. In [19], they applied optimization techniques to generate test cases, where the constraints are constructed based on MUTP, CUTPNFP, and MNFP strategies, and the objective is to minimize the test points to be selected. The results showed that optimization algorithm can generate a smaller test set than the heuristic one, but take more time to compute. This kind of approaches is limited by the capability of MUMCUT, and does not guarantee to obtain optimal solutions for detecting the double faults in the fault hierarchy. For detailed comparison with the proposed approach in this paper, see Section 10.6.

Lau et al. [27], [28], [29] studies detection conditions for double faults of Boolean expressions that may occur to terms and/or literals, in one term or two terms. They described that some test case selection strategies developed for the detection of single faults can also detect all double faults related to terms, but these strategies cannot guarantee to detect all double faults related to literals. Several supplementary strategies were developed to detect the double faults that cannot be revealed by the existing test strategies, including Pairwise Multiple Near False Points (PMNFP), Supplementary Multiple False Points (SMFP), Supplementary Pairwise Multiple False Point (SPMFP), Pairwise Multiple Unique True Point (PMUTP), Supplementary Multiple Unique True Point (SMUTP), and Supplementary Multiple Overlapping True Point (SMOTP). They did not provide approaches to minimize the test set while maintaining the detection capabilities of both single faults and double faults.

Fraser and Gargantini [12] presented a test suite generation using a SAT solver for test case generation, and it consists of a test predicate generator, a test suite generator and a SAT solver. However, the detection of faults is not guaranteed,

and the process may not produce optimal results [26]. Gargantini [13] suggested that one needs to consider the constraints among the variables contained in Boolean expressions. He pointed out that ignoring the constraints during test generation can reduce the fault detection capability of the tests. Recently, Arcaini and Gargantini [1] formalize an optimized SAT/SMT-based process with constraints over the inputs of Boolean expressions.

Tsai et al. [37] used K-map [22] representation of Boolean expressions to define their topological structure. The K-map allows systematic analysis on the boundaries and Hamming distances [16] in Boolean expressions as multiple Boolean expressions may represent the same topological structure in K-map, thus conceptually they should be treated as the same. They also proposed test case generation techniques, called "Swiss Cheese", based on K-map for testing service applications. Yu et al. explored the characteristics of topological structure of Boolean expressions and used heuristic approaches to generate test cases [39], [40].

Mutation testing proposed by DeMillo et al. [8] and Hamlet [15] requires tests that distinguish a program from a set of mutant programs. Kaminski et al. [21] proposed new mutation operators that use only subsuming higher order mutation operators. The approach is collectively called TRF-TIF (Term Reference Fault, Term Insertion Fault) that obtains the set of all test points to detect the nine types of faults or mutants in the fault hierarchy, and then selects a subset of test points from the large set to guarantee the detections of those mutants. TRF-TIF uses a locally optimal algorithm to obtain the test cases at each stage. Empirical analysis shows that TRF-TIF does not guarantee to produce an optimal solution, but can approximate a near-optimal solution in a reasonable time. Appendix D, available in the online supplemental material, further discusses TRF-TIF, and its relationship with a greedy algorithm. Section 10 compares the proposed approach with TRF-TIF.

3 Definitions and Notations

3.1 Primitive Definitions of Boolean Expressions

A Boolean expression S consists of a combination of Boolean variables and operators that produces either TRUE (1) or FALSE (0) when evaluated.

Boolean operators include AND, OR and NOT, denoted as "+", "·", "$^{-}$"[1] respectively, where "·" may be omitted. A literal is each occurrence of a Boolean variable a in an expression, either in positive form (e.g., a) or in negative form (e.g., $\bar{a}$). A term is a set of literals connected by these operators. When terms in a Boolean expression are separated by OR, and literals are separated by AND, it is called DNF. When terms in a Boolean expression are separated by AND, and literals are separated by OR, it is called Conjunctive Normal Form (CNF), e.g., $(a + b)(c + d)$. IDNF is a DNF such that none of its literals are redundant, and Irredundant CNF (ICNF) is a CNF such that none of its literals are redundant. While this paper is presented in terms of DNF, in all cases dual statements can be made about CNF.

1. Expressions in Appendixes A2 and A3, available in the online supplemental material, use "!" to represent NOT, as these expressions can be directly fed into the tool developed by the authors' team, to obtain the results.

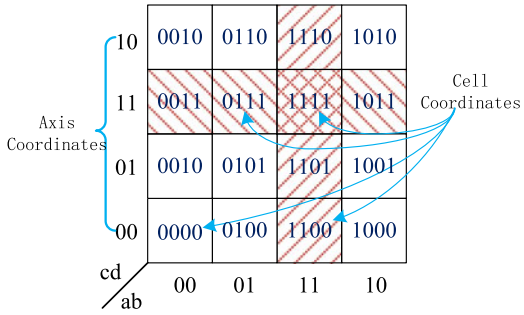


Fig. 2. $S_1 = ab + cd$ in K-map.

Boolean expression S is evaluated by an assignment, c, that maps each of Boolean variables, $c_i\,(i = 1, \ldots, n)$, in S to 1 or 0, denoted as $c = (c_1, \ldots, c_i, \ldots c_n)$, i.e., $S(c) = T_1(c) + \cdots + T_i(c) + \cdots + T_m(c)$, where $T_i(c)$ indicates the truth value of term T_i in S against an assignment c.

3.2 Basic Definitions in K-Map

K-map is a structural model for Boolean expressions [23], [39], [40]. A K-map corresponds to a truth table of a Boolean expression S with n variables. Each cell (a *min-term* in the literature) in K-map symbolizes a combination of values of n variables, called *a cell coordinate*, denoted as c. When a cell is selected to detect a type of faults, the cell is corresponding to a test case of S, i.e., an assignment of values of Boolean variables. Let $\mathcal{C}$ be a set of all the cell coordinates in K-map, then $c \in \mathcal{C}$. If $S(c) = 1$, i.e., the truth value is TRUE, then c is called a *true cell*; and if $S(c) = 0$, i.e., the truth value is FALSE, then c is called a *false cell*. Let $TC(S)$ denote the set of *true cells* of S and $FC(S)$ denote the set of *false cells* of S.

For a K-map with $\lceil \frac{n}{2} \rceil$-dimensions, when n is an even number and greater or equal to 4, the coordinates on each axis are represented with 2-bit Gray code [14], i.e., an *ordered* set {00, 01, 11, 10}, where the successive elements differ in only one bit; when n is equal to 1 or 2, the coordinates on each axis can be represented with a 1-bit Gray code, i.e., {0, 1}; and when n is an odd number and greater or equal to 3, only on the last dimension, Gray code is a 1-bit set, and other dimension(s) with 2-bit set. Without loss of generality, this paper illustrates the situation when n is an even number and greater than or equal to 4, i.e., on each dimension, Gray code is an *ordered* set {00, 01, 11, 10}.

The combination and orientation of Boolean variables on axes and the start point of axes do not affect the topological structure of a Boolean expression in K-map in terms of cell coordinates and their relative positions [39]. The $\lceil \frac{n}{2} \rceil$-dimensional grid is toroidally connected, that means that rectangular cells can wrap across the edges.

Example 2. Boolean expression $S_1 = ab + cd$ in Example 1 can be depicted in a $\lceil \frac{4}{2} \rceil$-*dimensional* (i.e., 2-dimensional) K-map as shown in Fig. 2, *where* x axis represents Boolean variables a and b, y axis represents variables c and d. The Gray code set {00, 01, 11, 10} is the axis *coordinates* on each dimension. Each cell in the K-map is *uniquely* identified by its cell coordinate. *Adjacent* cells on K-map have one and only one bit different, cells on the extreme right are actually *adjacent* or toroidally connected to those on the far left, e.g., {1011} and {0011}; similarly, so are those at

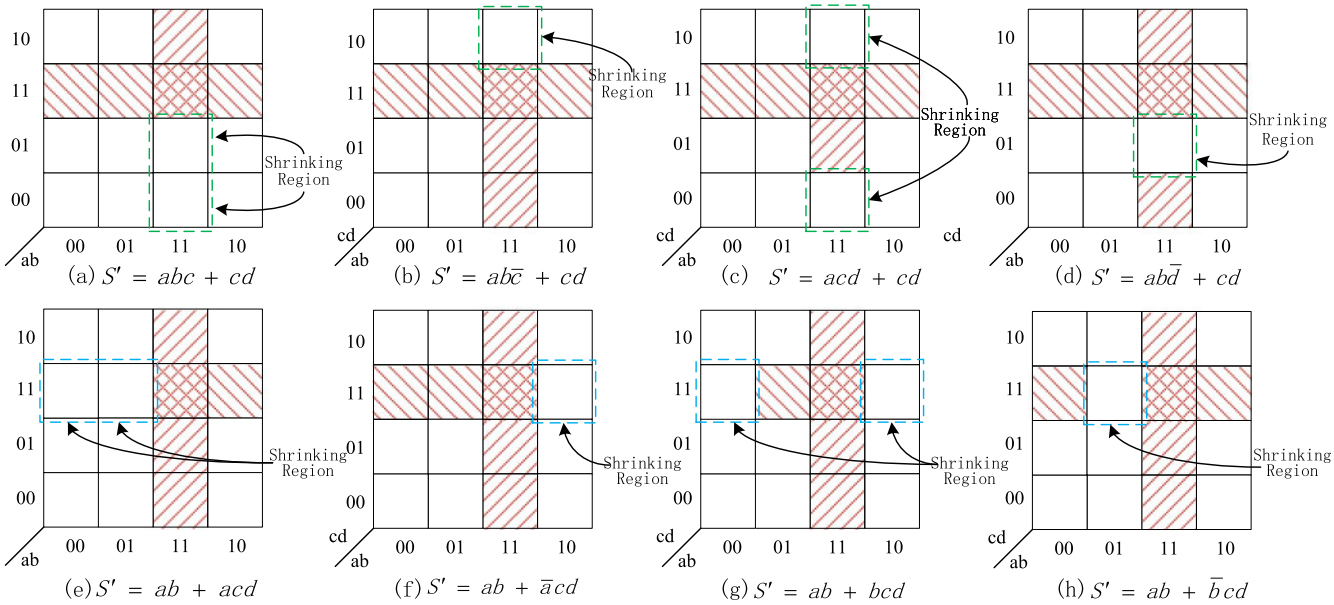


Fig. 3. Valid term and expression shrinking regions in K-map.

the top and those at the bottom, e.g., {0110} and {0100}. The shadowed cells in the K-map that make S_1 evaluated to 1 are true cells, and the other cells that make S_1 evaluated to 0 are false cells.

If $c \in TC(S) \wedge T_i\ (c) = 1 \wedge T_j\ (c) = 0, (i,\ j\ = 1, \ldots, m, i \neq j)$, c is called a *unique true cell* of T_i. If $c \in TC(S) \wedge T_i\ (c) = 1 \wedge T_j\ (c) = 1, (i, j\ = 1, \ldots, m, i \neq j)$, c is called a *shared true cell* (STC) of T_i and T_j. The *unique true cells* (UTC) of T_i can be obtained by taking out STCs with other terms from the true cells of T_i. UTC and STC correspond to UTP and *overlapping true point* (OTP) [43], respectively.

Example 3. In Fig. 2, {0011} is a unique true cell of term *cd*, i.e., $T_1(0011) = 0$, $T_2(0011) = 1$, and $S_1(0011) = 1$, indicating term *ab* is false, term *cd* is true, corresponding to the requirement in Example 1: "valve *A* is off and cabin temperature is less than *T* units, at the meantime, valve *C* is on and wind-force is greater than *W* units, therefore, the cruise parameter S_1 should be adjusted". {1111} is a STC of terms *ab* and *cd*, i.e., $T_1(1111) = 1$, $T_2(1111) = 1$, and $S_1(1111) = 1$; and {0000} is a false cell, i.e., $T_1(0000) = 1$, $T_2(0000) = 1$, and $S_1(0000) = 0$.

3.3 Regions in K-Map

A K-map is considered as a region, called *K-map region*, with 2nd cells that is closed, connected, and non-empty. In Fig. 2, all of the 16 cells in the K-map constitute a K-map region.

A Boolean expression S partitions a K-map region into true cell regions and false cell regions. The true cell regions are also called *Boolean expression regions*. The shadowed cells in Fig. 2 represent the region of $S_1 = ab + cd$.

Term region refers to all the true cells of T_i in S. Let α_i be the number of the literals that T_i consists of, then the term region contains $2^{n-\alpha_i}$ true cells.

Intuitively, Boolean expression faults, like strong external strengths, squeeze or/and stretch Boolean expression region, and incur the shrinking or/and expanding of topological structure of the expression in K-map from different dimensions to different terms.

4 Fault Topological Structures

This paper characterizes Boolean faults of Boolean expressions as the changes of topological structures in K-map.

4.1 Shrinking Regions

A shrinking region of term T_i is a set of true cells of T_i that make the faulty term evaluated as false due to an LIF. Given the number of variables, n, and the number of literals in T_i, α_i, the total possible number of shrinking regions of the term is $(n - \alpha_i) \times 2$, as for LIF, every variable that does not appear in T_i can be inserted in two forms, i.e., positive literal and negative literal. Given the number of terms in S, m, the total number of shrinking regions of S is $m \times (n - \alpha_i) \times 2$.

T_i may have STCs with other term(s). Taking out those STCs results in *valid shrinking regions of term* T_i. The union of valid shrinking regions of all terms in S is the *valid shrinking regions of Boolean expression* S.

Example 4. As for LIF to $T_1 = ab$ as in Fig. 2, there are four fault instances due to inserting literals $c, \bar{c}, d$, or $\bar{d}$ into *ab*, respectively. For example, inserting literal $\bar{d}$ into *ab* results in a shrinking region of *ab* with two cells, {1101, 1111} as shown in Fig. 3d, and these two cells make *ab* evaluated to true, and the faulty $ab\bar{d}$ evaluated to false. Cell {1111} is a STC with *cd*, therefore, the valid shrinking region due to the LIF is {1101} that guarantees to detect the fault, because $S(1101) = 1$ and $S'(1101) = 0$ where S' is the expression with the fault. The valid shrinking regions of *ab* consist of four regions with cells {1100, 1101}, {1110}, {1100, 1110} and {1101}, respectively as shown in Figs. 3a, 3b, 3c, and 3d; and likewise, the valid shrinking regions of *cd* consist of four regions with cells {0011, 0111}, {1011}, {0011, 1011} and {0111}, respectively as shown in Figs. 3e, 3f, 3g, 3h. The valid shrinking regions of the Boolean expression are the union of the valid shrinking regions of *ab* and the valid shrinking regions of *cd*.

In addition, ORF+ and TOF in the fault hierarchy in Fig. 1 also lead to shrinking regions.

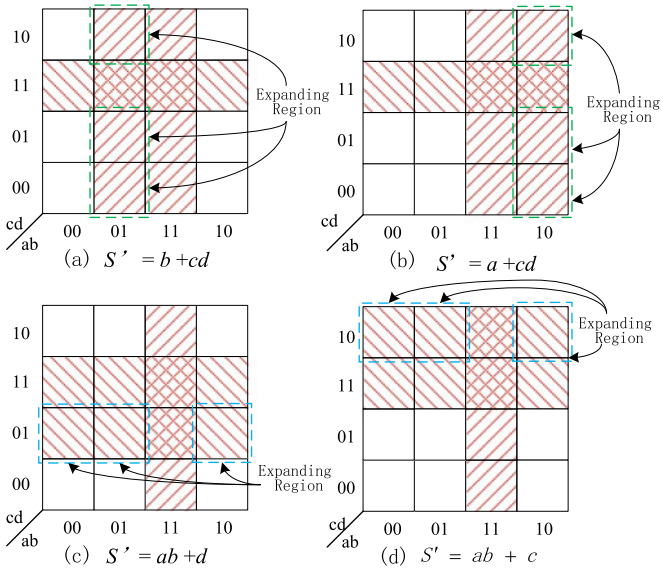


Fig. 4. Valid term and expression expanding regions in K-map.

4.2 Expanding Regions

An expanding region of term T_i is a set of surrounding false cells that make the faulty term evaluated to true due to an LOF. Given the number of literals in the term, α_i, the total possible number of expanding regions of the term is α_i, as for LOF, every literal of T_i can be omitted once. Given the number of terms in a Boolean expression S, m, the total number of expanding regions in S is $m \times \alpha_i$ that seems to have no appearance of n in the formula. Actually, n is the upper bound of α_i, i.e., $\alpha_i \leq n$, and $\alpha_i \geq 2$, because when $\alpha_i = 1$, it reduces to the Term Omission Fault (TOF).

The surrounding false cells of term T_i may be part of other term regions. Taking out those cells results in valid expanding regions of term T_i. The union of valid expanding regions of all terms in S is the *valid expanding regions of Boolean expression S*.

Example 5. As for LOF to term *ab* as in Fig. 2, there are two fault instances due to omitting the two literals *a* and *b* from *ab*, respectively. For example, omitting the literal *a* from *ab* results in an expanding region of *ab* with four cells, {0100, 0101, 0111, 0110} as shown in Fig. 4a, and these four cells make *ab* evaluated to false, and the faulty term *b* evaluated to true. Cell {0111} is a true cell of *cd*, therefore, the valid expanding region due to the LOF has three cells: {0100, 0101, 0110} that guarantees to detect the fault, because each of the cells makes S evaluated to 0 and S' to 1 where S' is the expression with the fault. The valid expanding regions of *ab* consist of two regions with cells {0100, 0101, 0110}, and {1000, 1001, 1010}, respectively as shown in Figs. 4a and 4b; and likewise, the valid expanding regions of *cd* consist of two regions with cells {0001, 0101, 1001}, {0010, 0110, 1010}, respectively as shown in Figs. 4c and 4d. The valid expanding regions of the Boolean expression are the union of the valid expanding regions of *ab* and the valid expanding regions of *cd*.

In addition, ORF· in the fault hierarchy in Fig. 1 also leads to expanding regions.

4.3 Shrinking and Expanding Regions

LRF will result in both shrinking and expanding regions.

A corresponding shrinking and expanding region of term T_i is a set of true and false cells that make the faulty term evaluated to false (corresponding to shrinking) and true (corresponding to expanding) due to an LRF. Given the number of variables, n, and the number of literals in T_i, α_i, the total possible number of corresponding shrinking and expanding regions of term T_i is $\alpha_i \times (n - \alpha_i) \times 2$, as for LRF, as every literal of the term can be replaced by each literal that does not appear in the term in two forms, i.e., positive literal and negative literal. Given the number of terms in Boolean expression S, m, the total number of corresponding shrinking and expanding regions of S is $m \times \alpha_i \times (n - \alpha_i) \times 2$.

T_i may have STCs with other term(s). Taking out those STCs results in *valid corresponding shrinking regions of term* T_i, up to the number of $(n - \alpha_i) \times 2$. The union of valid corresponding shrinking regions of all terms in S is the *valid corresponding shrinking regions of Boolean expression S*.

The surrounding false cells of T_i may be part of other term regions. Taking out those cells results in *valid corresponding expanding regions of term* T_i, up to $\alpha_i \times (n - \alpha_i) \times 2$. The union of valid corresponding expanding regions of all terms in S is the *valid corresponding expanding regions of Boolean expression S*.

Consequently, the union of valid corresponding shrinking and corresponding expanding regions of all terms in S is the *valid corresponding shrinking and expanding regions of Boolean expression S*.

Example 6. As for LRF to *ab* as in Fig. 2, there are eight fault instances due to replacing literals *a* and *b* of *ab* with literals $c, \bar{c}, d$, or $\bar{d}$, respectively. For example, faulty term *cb* is obtained by replacing the literal *a* in *ab* with literal *c*, that can be considered as inserting the literal *c* into *ab* and then omitting the literal *a*, where the former is a corresponding shrinking region of *ab* with cells {1100, 1101}, and the valid corresponding shrinking region is {1100, 1101} as shown in the lower middle part of Fig. 5a; the latter is a corresponding expanding region of *ab* with cells {0111, 0110}, and the valid corresponding expanding region is {0110} as shown in the upper left part of Fig. 5a; and accordingly, the corresponding shrinking and expanding region of the fault is composed of four cells {1100, 1101, 0111, 0110}, and the valid corresponding shrinking and expanding region of the fault has three cells {1100, 1101, 0110}.

Likewise, fault *ac* is replacing the literal *b* in *ab* with literal *c*, that can be considered as inserting the literal *c* into *ab* and then omitting the literal *b*, where the former is a corresponding shrinking of *ab* with cells {1100, 1101}, and the valid corresponding shrinking region is {1100, 1101} as shown in the lower middle part of Fig. 5b, and the latter is a corresponding expanding region of *ab* with cells {1011, 1010}, and the valid corresponding expanding region is {1010} as shown in the upper right part of Fig. 5b, and the valid corresponding shrinking and expanding region of the fault is composed of cells {1100, 1101, 1010}.

The corresponding shrinking region of *ab* due to an LRF instance in Fig. 5a is the same as the corresponding shrinking region of *ab* due to another LRF instance in Fig. 5b, as both insert literal *c* into *ab*. This is a very good characteristic of the topological structure of the

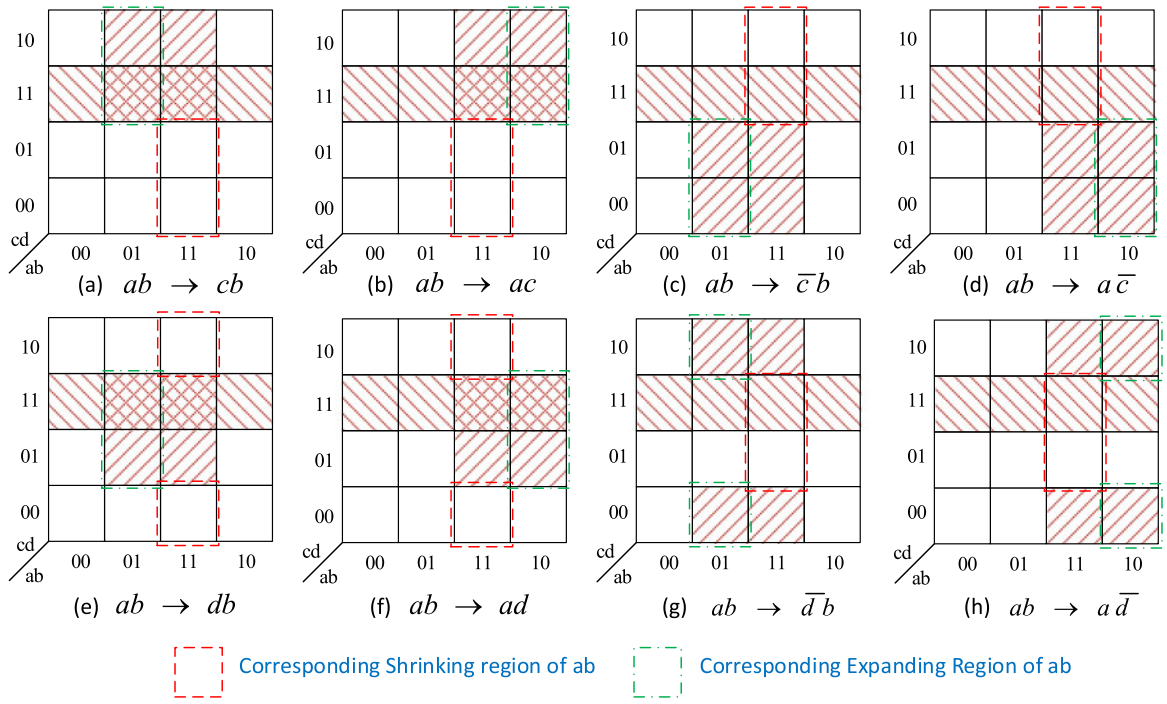


Fig. 5. Valid corresponding shrinking and expanding regions in K-map.

expression, as the two fault instances, cb and ac, can share one set of the valid shrinking region, and thus it saves the time to generate and solve the constraints to detect the other. Similarly, other six fault instances are shown in Figs. 5c, 5d, 5e, 5f, 5g, and 5h among them, $\bar{c}b$ and $a\bar{c}$ share the same valid corresponding shrinking region as shown in Figs. 5c and 5d, db and ad share the same valid corresponding shrinking region as shown in Figs. 5e, 5f, and $\bar{d}b$ and $a\bar{d}$ share the same valid corresponding shrinking region as shown in Figs. 5g and 5h. Theorem 4 in Appendix B5, available in the online supplemental material, provides formal proofs of these characteristics.

In addition, ENF, TNF and LNF in the fault hierarchy in Fig. 1 also lead to both shrinking regions and expanding regions.

4.4 Double-Faults and Fault Regions

The previous sections present the topological structures of single faults in the hierarchy. This section discusses the topological structures of double faults in the hierarchy, specifically, the eight double-faults that MUMCUT and minimal-MUMCUT cannot guarantee to detect, among them, double-faults to two different terms, including LIF $\bowtie^2$ LIF, LIF $\bowtie^2$ LRF, LRF $\bowtie^2$ LRF, TOF $\bowtie^2$ LRF, ORF+ $\bowtie^2$ LRF, where $\bowtie^2$ indicates double-faults to two different terms; and double-faults to a single term, including LIF $\bowtie^1$ LRF, LRF $\bowtie^1$ LRF, LOF $\bowtie^1$ LRF, where $\bowtie^1$ indicates double-faults to a single term.

Fig. 6 shows that an example of LIF $\bowtie^2$ LIF to two different terms, and it can be considered as shrinking twice in sequence, and eventually deciding the changes in the topological structure. Fig. 6a shows an example of Boolean expression, $S = ab + ac + d$, to which double faults will occur. Fig. 6b shows that there is a double-fault instance with two LIFs happened to two different terms: inserting literal c into ab, and leading the shrinking region consisting of cell {1100}, where cell {1101} is not a valid shrinking; inserting literal a into d, and leading the shrinking region consisting of cells {0001, 0101, 0011, 0111}. This double-fault instance can be detected by MUMCUT or minimal-MUMCUT. On the other hand, the double-fault instance arose in Fig. 6c cannot be detected by MUMCUT nor minimal-MUMCUT: cell {1110} is an overlapping point of terms ab and ac, and it is shrunk *twice* due to two LIFs occurred to each of the two terms, i.e., inserting $\bar{c}$ into ab, and inserting $\bar{b}$ into ac. This is because MUMCUT and minimal-MUMCUT never select such overlapping points, and nor the optimization model based on minimal-MUMCUT [19]. However, the topological structure of the Boolean expression is indeed changed, the *differential* is the double shrinking cell {1110}, and will be captured by the proposed cell-covering approach.

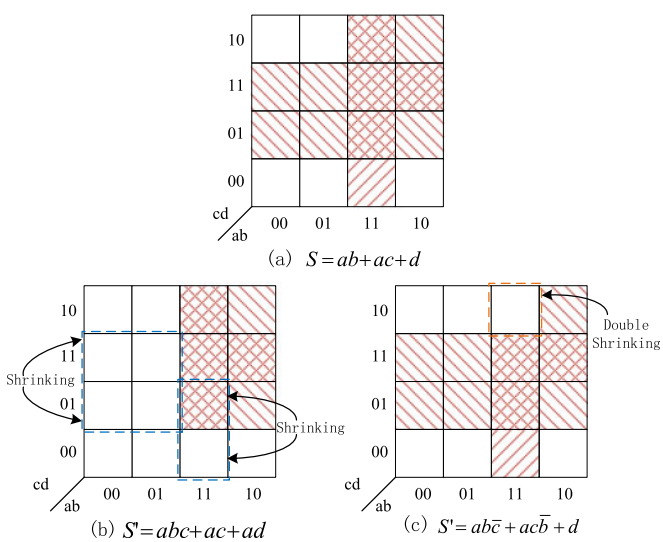


Fig. 6. Double faults LIF $\bowtie^2$ LIF in K-map.

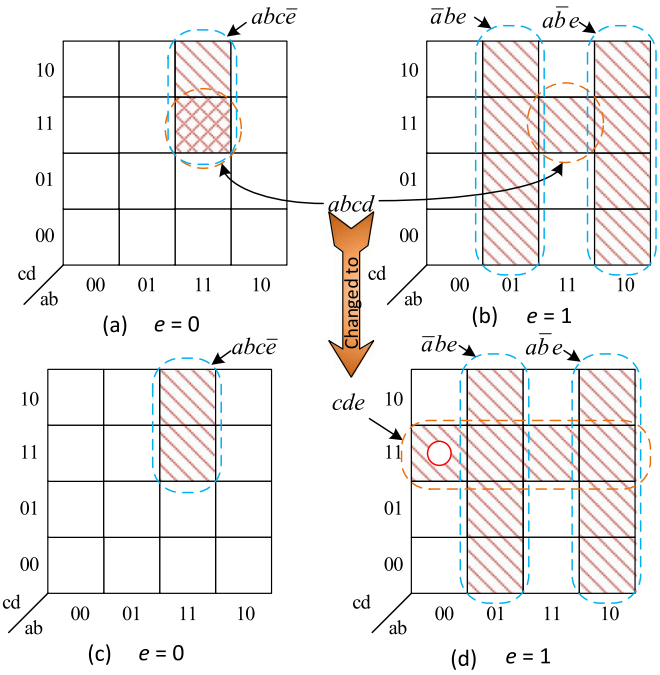


Fig. 7. Double-faults LIF ⋈ LRF in K-map.

Fig. 7 shows another example of double-faults LOF $\bowtie^1$ LRF. Figs. 7c and 7d where literal e takes 0 and 1, respectively, show that the fault happened to $abcd$ of expression $S = abcd + abc\bar{e} + \bar{a}be + a\bar{b}e$ in Figs. 7a and 7b where literal e takes 0 and 1, respectively. The double-fault can be considered as:

(1) LOF: Expand $abcd$ with cells {11110, 11111} to bcd with cells {11110, 11111, 01110, 01111} by omitting literal a; and
(2) Then LRF: one can think this way, shrinking bcd to $bcde$ with cells {11111, 01111} by inserting literal e into bcd, and then expanding $bcde$ to cde with cells {00111, 01111, 11111, 10111} by omitting literal b, i.e., replacing literal b with e.

Eventually, the double-fault leads to the change of the topological structure, one more true cell {00111} is added to true cell set of the original expression S. Cell {00111} is a non-NFP of $abcd$ and never be selected by MUMCUT or minimal-MUMCUT. Although it is a NFP of $\bar{a}be$ and $a\bar{b}e$, the two single faults occurred to these two terms can be detected without selecting cell {001111}. As a result, the double fault instance cannot be revealed by neither MUMCUT nor minimal-MUMCUT.

5 Fault-Detection Conditions

Once the topological structures of LIFs, LOFs and LRFs faults are identified, these faults can be detected if each of the valid regions has at least one cell in the selected test case set. Thus, fault detection can be characterized by cell covering in the K-map. This is stated as *a cell-covering of Boolean expression S*, and Appendix B1, available in the online supplemental material, provides the formal definition.

Different faults may have overlapping fault regions sharing same cells. For example, LIFs and LRFs in a term may share same shrinking regions, and LOFs and LRFs in a term may share same cells of their expanding regions. Different terms in a Boolean expression may share the same cells in their expanding regions as well. Identifying these shared cells can reduce the number of test cases while maintaining the fault detection capability. The following sections provide the conditions that guarantee to detect LIFs, LOFs and LRFs. See Appendix B2 to Appendix B6, available in the online supplemental material, for formal proofs.

5.1 Detection Condition for LIFs

A test set guarantees to detect LIFs if each of the valid shrinking regions of the expression has at least one cell in the selected test case set.

Let S' be the expression with T_i replaced by T_i' due to an LIF that partitions the region of T_i into two regions: a shrinking region and a remaining region of T_i, each with the same number of cells. The cells in both the shrinking region and the remaining region make T_i evaluated to true, consequently make S evaluated as true. The cells in the shrinking region make faulty T_i' evaluated to false, but this does not mean to make S' evaluated as false, because the cells in the shrinking region of T_i may be the STCs with T_j, making T_j evaluated as true, and thus S' evaluated as true. Therefore, the STCs in the shrinking region cannot differentiate S from S', i.e., not able to detect the LIF.

The valid shrinking regions are obtained by taking out such STCs from the shrinking regions, and any cells in the valid shrinking regions make S evaluated as true, S' as false, and thus able to detect the LIF. A valid shrinking region may be an empty set if the cells in a shrinking region are all STCs, i.e., the corresponding LIF of a term does not change the region of the expression in K-map, and does not incur any faults. Actually, the LIF is called an *equivalent mutant* in mutation testing [8]. Therefore, a valid shrinking region is required to be a non-empty set of cells, i.e., any cell in the valid shrinking region guarantees to the LIF.

Consequently, if each of the valid shrinking regions of S has at least one cell in the selected test set, the test set guarantees to detect all LIFs of S. For formal proofs, see **Theorem 1** in Appendix B2, available in the online supplemental material.

5.2 Detection Condition for LOFs

A test set guarantees to detect LOFs of S if each of the valid expanding regions of S has at least one cell in the selected test set.

Let S' be the expression with T_i replaced by T_i' due to an LOF that doubles the region of T_i and leads to two regions: an expanding region and the original region of T_i, each with the same number of cells. That is an LOF is related to an expanding region in K-map. The cells in the expanding region make faulty term T_i' evaluated to true, consequently make S' evaluated as true; and make the original T_i evaluated to false, but this does not mean to make S evaluated as false, because the cells in the expanding region of T_i may be in the region of T_j as well, even these cells make T_i evaluated to false, T_j is still evaluated as true, and thus S as true. Therefore, in this situation, the cells in the expanding region cannot differentiate S from S', i.e., not able to detect the LOF.

The valid expanding regions are obtained by taking out such true cells of other terms from the expanding regions, and any cells in the valid expanding regions make S evaluated as false, S' evaluated as true, and thus able to detect the

LOF. For S in IDNF, a valid expanding region cannot be an empty set, otherwise, the term can be further reduced by omitting the literal.

Consequently, if each of the valid expanding regions of S has at least one cell in the selected test set, the test set guarantees to detect all the LOFs of S. For formal proofs, see **Theorem 2** in Appendix B3, available in the online supplemental material.

5.3 Detection Condition for LRFs

A test set guarantees to detect LRFs of S if each of the valid corresponding shrinking and expanding regions of S has at least one cell in the selected test set.

Let S' be the expression with T_i replaced by T_i' due to an LRF that can be considered as follows: inserting a literal that does not appear in T_i in a positive form or in a negative form; and then omitting a literal that appears in T_i. Correspondingly, there is a shrinking region and a remaining region of T_i; and then there is an expanding region of the remaining region of T_i, among them, the first one is called a corresponding shrinking region, and the last one is called a corresponding expanding region; collectively called a corresponding shrinking and expanding region. That is an LRF is related to a corresponding shrinking and expanding region in K-map.

The cells in the corresponding shrinking region make T_i evaluated as true, consequently make S evaluated as true, and make faulty term T_i' evaluated as false, but this does not mean to make S' evaluated as false, because the cells in the corresponding shrinking region of T_i may be the STCs with T_j, making T_j evaluated as true, and thus S' as true. Therefore, in this situation, the cells in the corresponding shrinking region cannot differentiate S from S', i.e., not able to detect LRFs.

The valid corresponding shrinking regions are obtained by taking out such STCs from the corresponding shrinking regions, and any cells in the valid corresponding shrinking regions make S evaluated as true, S' as false, and thus able to detect the LRF.

The cells in the corresponding expanding region make faulty T_i' evaluated to true, consequently make S' evaluated as true; and make the original T_i evaluated to false, but this does not mean to make S evaluated as false, because the cells in the corresponding expanding region of T_i may be in the region of T_j as well, even these cells make T_i evaluated as false, T_j is still evaluated as true, and thus S as true. Therefore, in this situation, the cells in the corresponding expanding region cannot differentiate S from S', i.e., not able to detect LRFs.

The valid corresponding expanding regions are obtained by taking out such cells from the expanding regions, and any cells in the valid corresponding expanding regions make S evaluated as false, S' evaluated as true, and thus able to detect LRFs.

Consequently, if each of the valid corresponding shrinking and expanding regions of S has at least one cell in the selected test set, the test set guarantees to detect all LRFs of S. For formal proofs, see Theorem 3 in Appendix B4, available in the online supplemental material.

In fact, when a valid corresponding shrinking region is not empty, cells in the valid corresponding shrinking region alone can detect LRFs. The property is formally proved in Theorem 4 and Corollaries 1, 2 and 3 in Appendix B5, available in the online supplemental material.

Assume that l_{ij} is a literal that appears in T_i, and l_q is a literal that does not appear in T_i. If the valid corresponding expanding region of T_i due to l_{ij} replaced by l_q (i.e., an LRF instance) is not empty, cells in the valid corresponding expanding region guarantees to detect the LOF fault by omitting l_{ij} from T_i. The property is formally proved in Theorem 5 in Appendix B5, available in the online supplemental material.

5.4 Detection Condition for Double Faults

There exist subsuming relations among the double faults. For example, when the double faults occur to two different terms, LIF $\bowtie^2$ LRF subsumes TOF $\bowtie^2$ LRF, which in turn subsumes ORF+ $\bowtie^2$ LRF; LIF $\bowtie^2$ LIF subsumes LIF $\bowtie^2$ LRF only if the LIF has its valid shrinking regions; TOF $\bowtie^2$ LRF in turn subsumes ORF+ $\bowtie^2$ LRF; LIF $\bowtie^2$ LRF subsumes consecutively LRF $\bowtie^2$ LRF only if the LIF has its valid shrinking regions. When the double faults occur to a single term, LIF $\bowtie^1$ LRF subsumes LRF $\bowtie^1$ LRF only if the LIF has its valid shrinking regions; LRF $\bowtie^1$ LRF subsumes in turn LOF $\bowtie^1$ LRF only if the LRF has its valid expanding regions. In addition, LRF $\bowtie^1$ LRF subsumes LIF $\bowtie^2$ LRF only if the two shrinkings of LIF $\bowtie^2$ LRF do not shrink the overlapping points at one time.

6 Cell-Covering Problem

This section formulates a cell-covering problem to minimize the number of test cases for detecting LIFs, LOFs and LRFs.

6.1 Minimum Cell-Covering Problem

6.1.1 *Single Fault Minimum Cell-Covering Problem*

The minimum cell-covering problem is to find a smallest number of cells that cover the valid regions of a targeted fault f. Let C_S^f be a set of cells that contains all the cells in the valid regions of fault f in S, $\mathbb{R}_S^f$, i.e., $\mathrm{C}_S^f = \{c | \forall c \in r, \forall r \in \mathbb{R}_S^f\}$.

The minimum cell-covering problem can be formulated as an integer linear programming (ILP) problem.

Minimize the objective:

$$\mathbf{\mathit{f}} = \sum_{\mathrm{c} \in \mathrm{C}_\mathrm{S}^f} x_\mathrm{c} \tag{1}$$

Subject to the constraints:

$$\sum_{c \in r} x_c \geq 1 \ \forall r \in \mathbb{R}_S^f \tag{2}$$

$$x_c \in \{0, 1\} \ \forall c \in \mathrm{C}_S^f \tag{3}$$

Formula (1) minimizes the total number of cells, corresponding to the total number of test cases; formula (2) represents the constraints to cover every valid fault region associated with single faults of S, and "≥ 1" means at least one cell should be selected to detect the fault f in that valid fault region; and formula (3) is the 0-1 constraints such that every cell is either in the minimum cell-covering solution or not, i.e., selected or not selected.

6.1.2 Multi-Single Fault Minimum Cell-Covering Problem

The previous section addresses the minimum cell-covering problem for one fault type, i.e., only one type of faults in S. This section presents the formulation for multi-single fault $f \in \mathcal{F}$ where $\mathcal{F}$ refers to a set of fault types, i.e., to find a smallest set of cells that cover the given valid fault regions $\mathbb{R}_S^{\mathcal{F}}$. In this paper, multi-single faults refer to any of LIFs, LOFs and LRFs, but one and only one of them each time occurs to S. Let C_S^f be a set of cells that contains all the cells in valid regions of fault f in S, $\mathbb{R}_S^f$, i.e., $\mathrm{C}_S^f = \{c | \forall c \in r, \forall r \in \mathbb{R}_S^f, \forall f \in \mathcal{F}\}$.

The minimum cell-covering problem can be formulated as the following ILP.

Minimize the objective:

$$\mathbf{k} = \sum_{c \in \mathrm{C}_S^{\mathcal{F}}} x_c \tag{4}$$

Subject to the constraints:

$$\sum_{c \in r} x_c \geq 1 \ \forall r \in \mathbb{R}_S^f, \forall f \in \mathcal{F} \tag{5}$$

$$x_c \in \{0, 1\} \ \forall c \in \mathrm{C}_S^{\mathcal{F}}, \forall f \in \mathcal{F} \tag{6}$$

Formula (4) minimizes the total number of cells, corresponding to the total number of test cases; formula (5) represents the constraints to cover every valid fault region associated with multi-single faults of S, and "≥ 1" means at least one cell should be selected to detect any fault $f \in \mathcal{F}$ in that valid fault region; and formula (6) is the 0-1 constraints such that every cell is either in the minimum cell-covering solution or not, i.e., selected or not selected.

6.1.3 Multi-Double Fault Minimum Cell-Covering Problem

This section presents the formulation for multi-double fault $f \bowtie f$, $f \in \mathcal{F}$ where $\mathcal{F}$ refers to a set of fault types, i.e., to find a smallest set of cells that cover the given valid fault regions $\mathbb{R}_S^{\mathcal{F} \bowtie \mathcal{F}}$. In this paper, multi-double faults refer to any of five double faults double-faults to two different terms, i.e., LIF $\bowtie^2$ LIF, LIF $\bowtie^2$ LRF, LRF $\bowtie^2$ LRF, TOF $\bowtie^2$ LRF, ORF+ $\bowtie^2$ LRF; and three double-faults to a single term, i.e., LIF $\bowtie^1$ LRF, LRF $\bowtie^1$ LRF, LOF $\bowtie^1$ LRF, but one and only one of them each time occurs to S. Let $C_S^{f \bowtie f}$ be a set of cells that contains all the cells in valid regions of fault $f \bowtie f$ in Boolean expression S, $\mathbb{R}_S^{f \bowtie f}$, i.e., $C_S^{f \bowtie f} = \{c | \forall c \in r, \forall r \in \mathbb{R}_S^{f \bowtie f}, \forall f \in \mathcal{F}\}$.

The minimum cell-covering problem can be formulated as the following ILP.

Minimize the objective:

$$\mathbf{k} = \sum_{c \in C_S^{\mathcal{F} \bowtie \mathcal{F}}} x_c \tag{7}$$

Subject to the constraints:

$$\sum_{c \in r} x_c \geq 1 \ \forall r \in \mathbb{R}_S^{f \bowtie f}, \forall f \in \mathcal{F} \tag{8}$$

$$x_c \in \{0, 1\} \ \forall c \in C_S^{\mathcal{F} \bowtie \mathcal{F}}, \forall f \in \mathcal{F} \tag{9}$$

Formula (7) minimizes the total number of cells, corresponding to the total number of test cases; formula (8) represents the constraints to cover every valid fault region associated with multi-double faults of S, and "≥ 1" means at least one cell should be selected to detect any fault $f \bowtie f$, $f \in \mathcal{F}$ in that valid fault region; and formula (9) is the 0-1 constraints such that every cell is either in the minimum cell-covering solution or not, i.e., selected or not selected.

6.2 Generating Constraint Coefficient Matrices

The section presents algorithms for automatically generating constraint coefficient matrices for LIFs, LOFs and LRFs, respectively.

6.2.1 Generating Constraint Coefficient Matrices for LIFs

For formal description of the algorithm, see Appendix C2, available in the online supplemental material.

Inputs: the set of STCs of T_i ($i = 1, \ldots, m$); an ordered set of Gray codes that T_i projects to each dimension d of a K-map, $d \in D$, $D = \{1, \ldots, \lceil \frac{n}{2} \rceil, \text{when } n > 2; 1, n, \text{when } n \leq 2\}$.

Output: the constraint coefficient matrix for LIFs.

Step 1: Initialize the constraint coefficient matrix;

Step 2: Get a T_i of S;

Step 3: Work through each dimension d, and calculate all the valid shrinking regions at dimension d:

Case 1: when T_i has an ordered set of Gray codes with 4 elements at d, reduce the 4-elements set to adjacent or toroidally connected 2-elements set in 4 shrinking-ways. For each of the 4 shrinking-ways, the shrinking region is obtained by performing the product operations on the ordered set of Gray codes among all dimensions, where only on dimension d, using the reduced 2-elements set, while Gray codes on other dimensions remaining unchanged.

Case 2: when T_i has an ordered set of Gray codes with 2 elements at d, reduce the 2-elements set to 1-element set in 2 shrinking-ways. For each of the 2 shrinking-ways, the shrinking region is obtained by performing the product operations on the ordered set of Gray codes among all dimensions, where only on dimension d, using the reduced 1-element set, while Gray codes on other dimensions remaining unchanged.

Case 3: when T_i has an ordered set of Gray codes with 1 element at d, there is no shrinking region in this case.

In both Case 1 and Case 2, subtract the set of STCs of T_i from the shrinking region, and obtain the valid shrinking region; map the set of cell coordinates in the valid shrinking region to a row of the constraint matrix such that the coefficient is 1 if the cell is in the valid shrinking region, and 0 otherwise; and then add the row to the constraint coefficient matrix. Case 3 has no new row added to the constraint coefficient matrix.

Step 4: If there exists any untreated dimension, repeat Step 3;

Step 5: If there exists any untreated term, repeat from Step 2;

Step 6: Output the constraint coefficient matrix for LIFs.

Example 7. Fig. 8 shows $S = a + \bar{b}cd$ in K-map. With regard to LIFs, the cell-covering problem is formulated as follows:

Minimize the objective:

$$\ell = x_3 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{16} \quad \text{(Oshr-1)}$$

Subject to the constraints:

$$x_9 + x_{10} + x_{11} + x_{12} \geq 1 \quad \text{(shr-1)}$$

$$x_{13} + x_{14} + x_{16} \geq 1 \quad \text{(shr-2)}$$

$$x_9 + x_{10} + x_{13} + x_{14} \geq 1 \quad \text{(shr-3)}$$

$$x_{11} + x_{12} + x_{16} \geq 1 \quad \text{(shr-4)}$$

$$x_9 + x_{12} + x_{13} + x_{16} \geq 1 \quad \text{(shr-5)}$$

$$x_{10} + x_{11} + x_{14} \geq 1 \quad \text{(shr-6)}$$

$$x_3 \geq 1 \quad \text{(shr-7)}$$

$$x_c \in \{0,1\} \text{ for } \forall c \in \{3,\ 9,\ \ldots,\ 16\} \quad \text{(shr-8)}$$

Where constraints (shr-1) to (shr-6) cover the six valid shrinking regions of a, (shr-7) covers the only one valid shrinking region of $\bar{b}cd$, (shr-8) is the 0-1 constraints such that a cell is either in the minimum cell-covering solution or not, i.e., selected or not selected, and (O_{shr}-1) minimizes the number of the cells that appear in the constraints.

The common example inputs of the algorithms to generate coefficient matrices in this section include the following information: $n = 4$; $m = 2$; there are two dimensions, i.e., horizontal and vertical, where horizontal dimension associates literals a and b, and vertical dimension associates literals c and d; a has an ordered set {11,10} of Gray codes with 2 elements at the horizontal dimension, and an ordered set {00, 01, 11, 10} of Gray codes with 4 elements at the vertical dimension; $\bar{b}cd$ has an ordered set {00, 10} of Gray codes with 2 elements at the horizontal dimension, and an ordered set {11} of Gray codes with 1-element at the vertical dimension.

In addition to the common inputs, to generate the coefficient matrix for LIFs needs the set of STCs between a and $\bar{b}cd$, {1011}, as an input. Take a at the vertical dimension as an example:

In *Step 1*: initialize the constraint coefficient matrix, i.e., no entry in the matrix;

In *Step 2*: get a of S;

In *Step 3*: at the vertical dimension, as a has an ordered set {00,01,11,10} of Gray codes with 4 elements, therefore, reduce 4-elements set to 2-elements set in 4 shrinking-ways, i.e., {00, 01}, {01, 11}, {11, 10} or {10, 00}, where 2-elements {10, 00} are toroidally connected in K-map.

Consequently, the four shrinking regions are generated by performing the product operations between the ordered set {11,10} of Gray codes with 2-elements at the horizontal dimension of a and the 4 reduced Gray codes at the vertical dimension of a: {1100, 1101, 1000, 1001}, {1101, 1111, 1011, 1001}, {1111, 1110, 1011, 1010}, and {1110, 1100, 1010, 1000}.

Subtract the set of STC {1011} of T_i from the four shrinking regions, and obtain the four valid shrinking regions, {1100, 1101, 1000, 1001}, {1101, 1111, 1001}, {1111, 1110, 1010}, and {1110, 1100, 1010, 1000}; for each of the four valid shrinking regions, map the sets of cell coordinates in the valid shrinking region to a row of the constraint matrix.

For example, for the valid shrinking region {1111, 1110, 1010}, the row of constraint matrix is (0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1), where cells in the K-map are indexed from 1 to 16 as shown in Fig. 8, such that the coefficient entry of the row is 1 if the cell is in the valid shrinking region, and 0 otherwise; and then add the row to the constraint coefficient matrix. The row corresponds to the coefficient vector of the constraint (shr-4). The rest of three coefficient vectors of the constraints are (shr-3), (shr-5), and (shr-6).

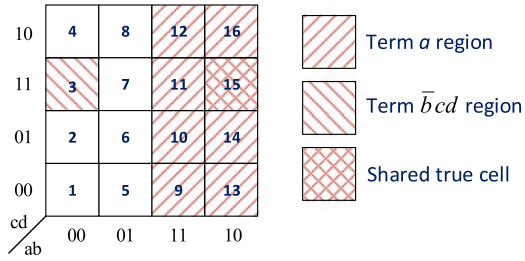


Fig. 8. An example of generating constraint coefficient matrices.

In *Step 4*: as there exists an untreated dimension on the horizontal, repeat the Step 3 on the horizontal dimension to obtain the two coefficient vectors of the constraints, (shr-1) and (shr-2);

In *Step 5*. as there exists untreated $\bar{b}cd$, repeat from Step 2 to obtain the one coefficient vectors of the constraints, (shr-7);

In *Step 6*: output the constraint coefficient matrix for LIFs.

6.2.2 Generating Constraint Coefficient Matrices for LOFs

For the formal description of the algorithm, see Appendix C3, available in the online supplemental material.

Inputs. the set of true cells of S; an ordered set of Gray codes that $T_i\ (i = 1, \ldots, m)$ projects to each dimension d of a K-map, $d \in D,\ D = \{1, \ldots, \lceil \frac{n}{2} \rceil, \text{when } n > 2; 1, n, \text{when } n \leq 2\}$.

Output: the constraint coefficient matrix for LOFs.

Step 1: Initialize the constraint coefficient matrix;

Step 2: Get a T_i of S;

Step 3: Work through each dimension d, and calculate all the valid expanding regions at dimension d:

Case 1: when T_i has an ordered set of Gray code with 1 element at d, increase the 1-element set to adjacent or toroidally connected 2-elements set in 2 expanding-ways. For each of the 2 expanding-ways, the expanding region is obtained by performing the product operations on the ordered set of Gray codes among all dimensions, where only on dimension d, using the increased 2-elements set, while Gray codes on other dimensions remaining unchanged.

Case 2: when T_i has an ordered set of Gray codes with 2 elements at d, increase the 2-elements set to 4-elements set in 1 expanding-way. The expanding region is obtained by performing the product operations on the ordered set of Gray codes among all dimensions, where only on dimension d, using the increased 4-elements set, while Gray codes on other dimensions remaining unchanged.

Case 3: when T_i has an ordered set of Gray codes with 4 elements at d, there is no expanding region in this case.

In both Case 1 and Case 2, subtract the set of true cells of T_i from the expanding region, and obtain the valid expanding region; map the set of cell coordinates in the

valid expanding region to a row of the constraint matrix such that the coefficient is 1 if the cell is in the valid expanding region, and 0 otherwise; and then add the row to the constraint coefficient matrix. Case 3 has no new row added to the constraint coefficient matrix.

Step 4: If there exists any untreated dimension, repeat Step 3;

Step 5: If there exists any untreated term, repeat from Step 2;

Step 6: Output the constraint coefficient matrix for LOFs.

Example 8. With regard to LOFs, the cell-covering problem for $S = a + \bar{b}cd$ in Fig. 8 is formulated as follows.

Minimize the objective:

$$\boldsymbol{k} = x_2 + x_4 + x_7 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{16} \quad \text{(Oexp-1)}$$

Subject to the constraints:

$$x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{16} \geq 1 \quad \text{(exp-1)}$$

$$x_7 \geq 1 \quad \text{(exp-2)}$$

$$x_2 \geq 1 \quad \text{(exp-3)}$$

$$x_4 \geq 1 \quad \text{(exp-4)}$$

$$x_c \in \{0,1\} \text{ for } \forall c \in \{2,\ 4,\ \ldots,\ 16\} \quad \text{(exp-5)}$$

Where constraint (exp-1) covers the only one valid expanding region of *a*, as the first term *a* has only one literal, and has no LOF but TOF; (exp-2) to (exp-4) cover the three valid expanding regions of $\bar{b}cd$, (exp-5) is the 0-1 constraints such that a cell is either in the minimum cell-covering solution or not, i.e., selected or not selected, and (O_{exp}-1) minimizes the number of the cells that appear in the constraints.

In addition to the common inputs, to generate the matrix needs the set of the true cells of the expression, {0011, 1100, 1101, 1111, 1110, 1000, 1001, 1011, 1010}, as an input. Take $\bar{b}cd$ at the vertical dimension as an example.

In *Step 1*: initialize the constraint coefficient matrix, i.e., no entry in the matrix;

In *Step 2*: get the first term *a* of *S*;

In *Step 2*: at the vertical dimension, as $\bar{b}cd$ has an ordered set {11} of Gray code with 1 element, therefore, increase the 1-element set to 2-elements set in two expanding-ways, i.e., {01,11} or {11,10}. Consequently, the two expanding regions are generated by performing the product operations between the ordered set {00,10} of Gray codes with 2 elements at the horizontal dimension of $\bar{b}cd$ and the 2 increased Gray codes {01} and {10} at the vertical dimension of $\bar{b}cd$: {0001,1001}, {0010,1010}. Subtract the set of true cells from the two expanding regions, and obtain the two valid expanding regions, {0001}, {0010}; for each of the two valid expanding regions, map the set of cell coordinates in the valid expanding region to a row of the constraint matrix. For example, for the valid expanding region {0001}, the row of constraint matrix is (0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0), where cells in the K-map are indexed from 1 to 16 as shown in Fig. 8, such that the coefficient entry of the row is 1 if the cell is in the valid expanding region, and 0 otherwise; and then add the row to the constraint coefficient matrix. The row corresponds to the coefficient vector of the constraint (exp-3). Similarly, the other coefficient vector of the constraints is (exp-4).

In *Step 4*: as there exists an untreated dimension on the horizontal, repeat the Step 3 on the horizontal dimension to obtain the coefficient vector of the constraints (exp-2);

In *Step 5*: as there exists untreated *a*, repeat from Step 2 to obtain the only coefficient vector of the constraints (exp-1);

In *Step 6*: output the constraint coefficient matrix for LOFs.

6.2.3 Generating Constraint Coefficient Matrices for LRFs

For a formal description of the algorithm, see Appendix C4, available in the online supplemental material.

Inputs: the set of STCs of T_i ($i = 1, \ldots, m$); the set of true cells of *S*; and an ordered set of Gray codes that T_i projects to each dimension d of a K-map, $d \in D,\ D = \{1, \ldots, \lceil \frac{n}{2} \rceil,$ when $n > 2; 1, n,$ when $n \leq 2\}$.

Output: the constraint coefficient matrix for LRFs.

Step 1: Initialize the constraint coefficient matrix;

Step 2: Get a T_i of *S*;

Step 3: Work through each dimension d, and calculate the valid corresponding shrinking regions and the valid corresponding expanding regions at dimension d.

Step 3.1 Calculate the valid corresponding shrinking regions at dimension d:

Case 1: when T_i has an ordered set of Gray codes with 4 elements at d, reduce the 4-elements set to adjacent or toroidally connected 2-elements set in 4 shrinking-ways. For each of the 4 shrinking-ways, the corresponding shrinking region is obtained by performing the product operations on the ordered set of Gray code among all dimensions, where only on dimension d, using the reduced 2-elements set, while Gray codes on other dimensions remaining unchanged.

Case 2: when T_i has an ordered set of Gray codes with 2 elements at d, reduce the 2-elements set to 1-element set in 2 shrinking-ways. For each of the 2 shrinking-ways, the corresponding shrinking region is obtained by performing the product operations on the ordered set of Gray codes among all dimensions, where only on dimension d, using the reduced 1-element set, while Gray codes on other dimensions remaining unchanged.

Case 3: when T_i has an ordered set of Gray codes with 1 element at d, there is no corresponding shrinking region in this case.

In both case 1 and case 2, subtract the set of STCs of T_i from the corresponding shrinking regions; and if the obtained valid corresponding shrinking regions are not empty, map the set of cell coordinates in the valid corresponding shrinking regions to a row of the constraint matrix such that the coefficient is 1 if the cell is in the valid corresponding shrinking region, and 0 otherwise, add the row to the constraint coefficient matrix.

Step 3.2: for each of the corresponding shrinking regions that the valid corresponding shrinking regions are empty, calculate the valid corresponding expanding regions at dimension d' other than d in Step 3.1:

Case 1: when term T_i has an ordered set of Gray code with 1 element at d', increase the 1-element set to toroidally

connected 2-elements set in 2 expanding-ways. For each of the 2 expanding-ways, the corresponding expanding region is obtained by performing the product operations on the ordered set of Gray codes among all dimensions, where on dimension d, using the set of Gray codes in the remaining region, while Gray codes on other dimensions using the increased 1-element set.

Case 2: when the remaining region of T_i has an ordered set of Gray codes with 2 elements at d', increase the 2-elements set to 4-elements set in 1 expanding-way. The corresponding expanding region is obtained by performing the product operations on the ordered set of Gray codes among all dimensions, where on dimension d, using the set of Gray codes in the remaining region, while Gray codes on other dimensions using the increased 2-elements set.

Case 3: when T_i has an ordered set of Gray codes with 4 elements at d', there is no corresponding expanding region in this case.

In both Case 1 and Case 2, subtract the set of true cells of T_i from the expanding region, and obtain the valid corresponding expanding region; map the set of cell coordinates in the valid corresponding expanding region to a row of the constraint matrix such that the coefficient is 1 if the cell is in the valid expanding region, and 0 otherwise; and then add the row to the constraint coefficient matrix. According to Theorem 5 in the Appendix B5, available in the online supplemental material, when the valid corresponding expanding region is not empty, cells in the valid corresponding expanding region that guarantee to detect the LRF guarantee to detect the LOF; therefore, the row of the constraint matrix for LOF can be omitted, saving the computation time. Case 3 has no new row added to the constraint coefficient matrix.

Step 4: If there exists any dimension, repeat from Step 3;

Step 5: If there exists any untreated term, repeat from Step 2;

Step 6: Output coefficient matrix for LRFs.

Example 9. With regard to LRFs, the cell-covering problem for $S = a + \bar{b}cd$ in Fig. 8 is formulated as follows.

Minimize the objective:

$$\text{Ocse} = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{16} \quad \text{(Ocse-1)}$$

Subject to the constraints:

$$x_5 + x_6 + x_7 + x_8 + x_{13} + x_{14} + x_{16} \geq 1 \quad \text{(cse-1)}$$

$$x_1 + x_2 + x_4 + x_9 + x_{10} + x_{11} + x_{12} \geq 1 \quad \text{(cse-2)}$$

$$x_4 + x_7 + x_8 + x_9 + x_{10} + x_{13} + x_{14} \geq 1 \quad \text{(cse-3)}$$

$$x_1 + x_2 + x_5 + x_6 + x_{11} + x_{12} + x_{16} \geq 1 \quad \text{(cse-4)}$$

$$x_2 + x_6 + x_7 + x_9 + x_{12} + x_{13} + x_{16} \geq 1 \quad \text{(cse-5)}$$

$$x_1 + x_4 + x_5 + x_8 + x_{10} + x_{11} + x_{14} \geq 1 \quad \text{(cse-6)}$$

$$x_7 \geq 1 \quad \text{(cse-7)}$$

$$x_2 \geq 1 \quad \text{(cse-8)}$$

$$x_4 \geq 1 \quad \text{(cse-9)}$$

$$x_3 \geq 1 \quad \text{(cse-10)}$$

$$x_c \in \{0,1\} \text{ for } \forall c \in \{1,2,\ldots,16\} \quad \text{(cse-11)}$$

Where constraints (cse-1) to (cse-6) cover the six valid corresponding shrinking and expanding regions of a, (cse-7) to (cse-10) cover the four valid corresponding shrinking and expanding region of $\bar{b}cd$, (cse-11) is the 0-1 constraints such that a cell is either in the minimum cell-covering solution or not, i.e., selected or not selected, and (O_{cse}-1) minimizes the number of the cells that appear in the constraints.

In addition to the common inputs, to generate the matrix needs the set of true cells of the expression {0011, 1100, 1101, 1111, 1110, 1000, 1001, 1011, 1010} and the set of the STCs between a and $\bar{b}cd$ {1011} as inputs. Take $\bar{b}cd$ at the horizontal dimension as an example:

In *Step 1*: initialize the constraint coefficient matrix;

In *Step 2*: get the second term $\bar{b}cd$;

In *Step 3*: at the horizontal dimension d calculate the valid corresponding shrinking region and the valid corresponding expanding regions.

In *Step 3.1*: Calculate the valid corresponding shrinking regions.

Because $\bar{b}cd$ has an ordered set {00, 10} of Gray codes with 2 elements at the horizontal dimension, therefore, reduce 2-elements set to 1-element set in 2 shrinking-ways, i.e., {00}, or {10}. Consequently, the two corresponding shrinking regions are generated by performing the product operations between the 2 reduced Gray codes at the horizontal dimension of $\bar{b}cd$ and the ordered set {11} of Gray code with 1 element at the vertical dimension of $\bar{b}cd$ {0011}, {1011}. Subtract the set of STCs {1011} of a and $\bar{b}cd$ from the two corresponding shrinking regions, and obtain the one valid corresponding shrinking region {0011}; map the set of the cell coordinate in the valid corresponding shrinking region to a row of the constraint matrix, i.e., (0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0), where cells in the K-map are indexed from 1 to 16 as shown in Fig. 8, such that the coefficient entry of the row is 1 if the cell is in the valid corresponding shrinking region, and 0 otherwise; and then add the row to the constraint coefficient matrix. The row corresponds to the coefficient vector of the constraint (cse-10).

Step 3.2: Because the valid corresponding shrinking region of {1011} is empty, calculate the corresponding expanding regions, where d' is the vertical dimension, and d is the horizontal dimension.

Because $\bar{b}cd$ has an ordered set {11} of Gray codes with 1 element at the vertical dimension d', therefore, increase the 1-element set to 2-elements set in two expanding-ways, i.e., {01, 11} or {11, 10}. Consequently, the two corresponding expanding regions are generated by performing the product operations between the ordered set {00} of Gray code of the remaining region of $\bar{b}cd$ with 1 element at the horizontal dimension and the 2 increased Gray codes {01} and {10} at the vertical dimension of $\bar{b}cd$: {0001}, {0010}. Subtract the set of true cells from the two corresponding expanding regions, and the two valid corresponding expanding regions are still {0001} and {0010}; for each of the two valid corresponding expanding regions, map the set of cell coordinates in the valid corresponding expanding region to two rows of the

constraint matrix. For example, for the valid corresponding expanding region {0001}, the row of constraint matrix is (0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0), where cells in the K-map are indexed from 1 to 16 as shown in Fig. 8, such that the coefficient entry of the row is 1 if the cell is in the valid expanding region, and 0 otherwise; and then add the row to the constraint coefficient matrix. The row corresponds to the coefficient vector of the constraint (cse-8). The other coefficient vector of the constraints is (cse-10).

In *Step 4*: as there exists an untreated dimension on the horizontal, repeat the Step 3 on the horizontal dimension to obtain the two coefficient vectors of the constraints, (cse-7) and (cse-9);

In *Step 5*: as there exists untreated *a*, repeat from Step 2 to obtain the six coefficient vectors of the constraints, (cse-1) to (cse-6);

In *Step 6*: output the constraint coefficient matrix for LIFs.

Note that the valid corresponding expanding region {0001} of $\bar{b}cd$ is not empty, the cell guarantees to detect the LRF by replacing literal *c* with $\bar{a}$, and also can detect the LOF by omitting literal *c*, i.e., there is no need to generate the row of the constraint matrix for the LOF. The fact can be found in the Example 8 and Example 9 for LOF and LRF in Sections 6.2.2 and 6.2.3, respectively, where the two same rows of the constraint matrix are obtained, i.e., (0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0).

7 ILP Optimization

The ILP problem can be optimized in at least two ways: 1) reducing the ILP constraint coefficient matrix by exploring the relationships among LIFs, LOFs, and LRFs; and 2) decomposing the ILP problems into independent sub-problems and solve each sub-problem first.

7.1 Generating Constraint Coefficient Matrices for LIFs, LOFs and LRFs

Relationship Between LIFs & LRFs. LIFs and LRFs may overlap as LIFs involve shrinking regions, and LRFs involve both shrinking and expanding regions. If the corresponding shrinking region of an LRF is covered by the shrinking region of an LIF, it is not necessary to develop test cases for the LRF. This is formally proved Theorem 4 in Appendix B5, available in the online supplemental material.

Relationship Between LRFs & LOFs. If the valid corresponding expanding region of LRFs is not empty, and it is a subset of the expanding region of an LOF, thus once one has the test cases from the valid corresponding expanding regions for the LRF, it is not necessary to develop the test cases for the LOF. This is formally described as Theorem 5 in Appendix B5, available in the online supplemental material.

These two properties have been identified earlier using a different approach [20]. Our contribution is a topological interpretation and uses these properties in reducing the constraint coefficient matrix to solve the ILP.

These two properties also lead to an interesting phenomenon: one may use less time to cover LIFs, LOFs, and LRFs than if one needs to cover LRFs only.

An updated algorithm to *generate the* constraint coefficient matrix *for LIFs, LOFs and LRFs is given below:*

Inputs: the set of STCs of $T_i\ (i = 1, \ldots, m)$; the set of true cells of *S*; an ordered set of Gray codes that T_i projects to each dimension d of a K-map, $d \in D$, $D = \{1, \ldots, \lceil \frac{n}{2} \rceil$, when $n > 2; 1, n$, when $n \leq 2\}$.

Output: the constraint coefficient matrix for LIFs, LOFs and LRFs.

Step 1: Produce the valid shrinking region(s) and generate the constraint coefficient row(s) for LIFs on each of the dimensions in K-map as described in Section 6.2.1; if the valid shrinking region(s) is/are empty, produce the valid corresponding expanding region(s) and generate the constraint coefficient row(s) for LRFs on the dimension in K-map as described in Section 6.2.3 and end Step 1, otherwise produce the valid expanding region(s) and generate the constraint coefficient row(s) for LOFs on the dimension in K-map as described in Section 6.2.2.

Step 2: Output the constraint coefficient matrix for LIFs, LOFs and LRFs.

Example 10. With regard to LIFs, LOFs and LRFs, the cell-covering problem of $S = a + \bar{b}cd$ in Fig. 8 is formulated as follows:

Minimize the objective:

$$\text{Min} = x_2 + x_3 + x_4 + x_7 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{16} \quad \text{(Oior-1)}$$

Subject to the constraints:

$$x_9 + x_{10} + x_{11} + x_{12} \geq 1 \quad \text{(ior-1)}$$

$$x_{13} + x_{14} + x_{16} \geq 1 \quad \text{(ior-2)}$$

$$x_9 + x_{10} + x_{13} + x_{14} \geq 1 \quad \text{(ior-3)}$$

$$x_{11} + x_{12} + x_{16} \geq 1 \quad \text{(ior-4)}$$

$$x_9 + x_{12} + x_{13} + x_{16} \geq 1 \quad \text{(ior-5)}$$

$$x_{10} + x_{11} + x_{14} \geq 1 \quad \text{(ior-6)}$$

$$x_3 \geq 1 \quad \text{(ior-7)}$$

$$x_7 \geq 1 \quad \text{(ior-8)}$$

$$x_2 \geq 1 \quad \text{(ior-9)}$$

$$x_4 \geq 1 \quad \text{(ior-10)}$$

$$x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{16} \geq 1 \quad \text{(ior-11)}$$

$$x_c \in \{0, 1\} \text{ for } \forall c \in \{2, 3, \ldots, 16\} \quad \text{(ior-12)}$$

Where constraints (ior-1) to (ior-7) come from (shr-1) to (shr-7), (ior-8) to (ior-10) from (cse-7) to (cse-9), and (ior-11) from (exp-1), (ior-12) is the 0-1 constraints such that a cell is either in the minimum cell-covering solution or not, i.e., selected or not selected, and (O_{ior}-1) minimizes the number of the cells that appear in the constraints.

The seven constraints (cse-1) to (cse-6) and (cse-10) that are necessary to detect LRF in Section 6.2.3 do not appear in the constraints that guarantee to detect the three faults, LIF, LOF and LRF. This is because satisfying the constraints (shr-1) to (shr-7) of LIFs implies satisfying the constraints (cse-1) to (cse-6) and (cse-10), the former goes to the multiple-faults constraints, while the latter does not. For example, constraint (shr-2) $x_{13} + x_{14} + x_{16} \geq 1$ implies constraint (cse-1) $x_5 + x_6 + x_7 + x_8 + x_{13} + x_{14} + x_{16} \geq 1$, and therefore, the former appears in the multiple single-fault constraints, while the latter does not.

As a result, three variables (x_5, x_6, x_8) and one constraint ($x_5 + x_6 + x_7 + x_8 + x_{13} + x_{14} + x_{16} \geq 1$) are completely eliminated in the ILP. This means that by applying the relationships between LIFs and LRFs, LRFs and LOFs, not only the number of constraints is reduced, but also the number of variables in the problem is reduced. This impacts the size of the problem dramatically reduced, and leads the time needed to solve the ILP problem reduced consequently.

7.2 Generating Constraint Coefficient Matrices for Multi-Double Faults

Relationships Among Double Faults in a Single Term. The three double-faults LIF $\bowtie^1$ LRF, LRF $\bowtie^1$ LRF and LOF $\bowtie^1$ LRF all have LRF and can apply its topological structure properties. In addition, the topological structure relationships between LIF and LRF and between LRF and LOF can be used as well when automatically generating the constraints coefficient matrices.

Relationship Among Double Faults in Two Different Terms. As LIF $\bowtie^2$ LRF subsumes TOF $\bowtie^2$ LRF that in turn subsumes ORF+ $\bowtie^2$ LRF, only three double-faults LIF $\bowtie^2$ LIF, LIF $\bowtie^2$ LRF and LRF $\bowtie^2$ LRF need to be handled. Again the topological structure properties of LIF and LRF can be applied such that the number of constraints of ILP can be reduced, saving the time to solve the problem.

Relationship Between LRF $\bowtie^1$ LRF and LIF $\bowtie^2$ LRF. The subsumption can be described as follows: the LRF $\bowtie^1$ LRF subsumes the LIF $\bowtie^2$ LRF only if the two shrinkings due to LIF $\bowtie^2$ LRF do not shrink the overlapping points of the two different terms at one time.

The algorithm to *generate the* constraint coefficient matrix *for the double faults is given below:*

Inputs: the set of STCs of two terms T_i, T_j $(i, j = 1, \ldots, m, i \neq j)$, and the set of rest STCs of T_i $(i = 1, \ldots, m)$; the set of true cells of S; an ordered set of Gray codes that T_i projects to each dimension d of a K-map, $d \in D$, D $= \{1, \ldots, \lceil \frac{n}{2} \rceil$, when $n > 2; 1, \text{n},$ when $n \leq 2\}$.

Output: the constraint coefficient matrix for multi-double faults.

Step 1: Produce the valid shrinking and expanding regions for LIF $\bowtie^1$ LRF on each of the dimensions in K-map and generate the constraint coefficient row(s); if both the valid regions are empty, produce the valid corresponding expanding region(s) for LRF $\bowtie^1$ LRF on the dimension in K-map and generate the constraint coefficient row(s) and end Step 1, otherwise produce the valid expanding region(s) LOF $\bowtie^1$ LRF on the dimension in K-map and generate the constraint coefficient row(s);

Step 2: Produce the valid shrinking regions for LIF $\bowtie^2$ LIF on each of the dimensions in K-map and generate the constraint coefficient row(s); if the valid shrinking regions are empty and the shared cells of two terms are shrunk due to LIF $\bowtie^2$ LRF, produce the valid corresponding expanding region(s) for the LIF $\bowtie^2$ LRF on the dimension in K-map and generate the constraint coefficient row(s) and end Step 2, otherwise produce the valid expanding region(s) LRF $\bowtie^2$ LRF on the dimension in K-map and generate the constraint coefficient row(s);

Step 3: Output the constraint coefficient matrix for multi-double faults.

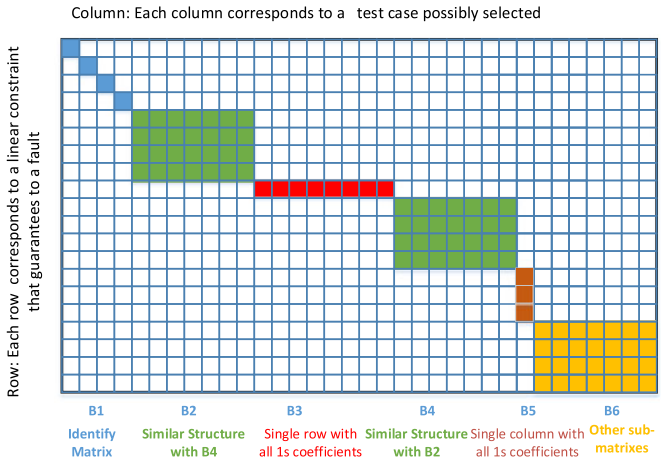


Fig. 9. Typical matrix after diagonal transformation.

7.3 Optimization by Diagonalization

Once a constraint coefficient matrix is obtained, one can perform optimization to speed up ILP execution by exchanging rows or columns so that elements in the matrix will reside on the diagonal of the matrix only. Mathematically, this is equivalent to changing indices of variables, as long as the indices are changed consistently, the modified matrix represents the same problem as the original matrix. However, interesting problems can now be observed as blocks on the diagonal can have the following forms:

- *Identity matrix*: Constraints in this block are represented as $x_i >= 1, \ldots, x_{i'} >= 1$, where $i \neq i'$ implying that each cell can detect only one fault, and each fault can be detected by only one cell.
- *Row matrix*: There is only one constraint in this block: $x_j + x_{j+1} + \ldots x_{j+k} >= 1$, implying any test case from column *j*th to column *k*th can detect the fault.
- *Column matrix*: Rows in this block have the same constraint: $x_r >= 1$, implying that one test case can detect all the faults associated with these rows.
- *Identical blocks*: At least there are two blocks on the diagonal having the same structure. It indicates that the sub-problems corresponding to these blocks have identical mathematical structure except for different elements participating. Solving one sub-problem will also solve the other sub-problems.

Fig. 9 illustrates the typical results of a transformation, where each column corresponds to a test case possibly selected, and each row corresponds to a linear constraint that guarantees to a fault. There are six blocks on the diagonal line, B1 through B6.

B1 is an identity matrix, B3 has only one row, and B5 has only one column. Thus, sub-problem B1, B3 and B5 are trivial sub-problems and can be easily solved. B2 and B4 have identical structures, thus only one run of ILP will be sufficient. B6 is a unique block and needs to be solved by ILP, however the matrix for B6 is much smaller than the original matrix, and thus ILP execution will be much more efficient.

Transforming the original matrix of ILP into a diagonal matrix will significantly reduce computation time, and any decreasing in size will lead to significant reductions in computation.

TABLE 1
The Impacts of the Number of Fractional Values and the Sub-Problems on Time Comsuptions in 20-Case

Exp. IDs	# of variables	# of terms	Lengths of Terms	# of occurrences	# of sub-problems	# of fractional values	Optimal/ Approximate	Time (Milliseconds)
1	7	5	5 - 5.80 - 6	2 - 4.14 - 5	2 / 10 / 1	0	Optimal	16
2	9	13	8 - 8.00 - 8	7 - 11.56 - 13	3 / 33 / 7	0	Optimal	31
3	12	25	4 - 5.84 - 7	1 - 12.17 - 25	22 / 0 / 10	83	Optimal	1,141
4	5	3	1 - 2.33 - 3	1 - 1.40 - 2	3 / 3 / 0	0	Optimal	16
5	9	9	1 - 3.11 - 4	1 - 3.11 - 8	5 / 4 / 2	20	Optimal	62
6	11	6	9 - 9.67 - 11	2 - 5.27 - 6	3 / 29 / 7	0	Optimal	70
7	10	8	7 - 7.50 - 8	4 - 6.00 - 8	4 / 4 / 12	0	Optimal	31
8	8	4	8 - 8.00 - 8	4 - 4.00 - 4	0 / 1 / 0	0	Optimal	62
9	7	2	7 - 7.00 - 7	2 - 2.00 - 2	0 / 1 / 0	0	Optimal	50
10	13	6	10 - 10.00 -10	2 - 4.62 - 6	5 / 0 / 19	0	Optimal	62
11	13	9	6 - 7.00 - 8	3 - 4.85 - 9	14 / 3 / 5	0	Optimal	172
12*	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
13	12	6	1 - 2.33 - 5	1 - 1.17 - 2	5 / 0 / 2	0	Optimal	313
14	7	6	2 - 2.67 - 3	1 - 2.29 - 4	5 / 0 / 2	4	Optimal	32
15	9	11	2 - 2.91 - 3	1 - 3.56 - 8	6 / 1 / 5	13	Optimal	78
16	12	23	2 - 3.78 - 5	1 - 7.25 - 16	16 / 0 / 8	24	Optimal	1,016
17	11	6	4 - 5.33 - 6	2 - 2.91 - 6	9 / 0 / 2	10	Optimal	62
18	10	8	4 - 4.75 - 5	2 - 3.80 - 8	9 / 0 / 2	23	Optimal	141
19	8	4	5 - 5.00 - 5	2 - 2.50 - 4	5 / 0 / 5	0	Optimal	31
20	7	2	6 - 6.00 - 6	1 - 1.71 - 2	1 / 8 / 1	0	Optimal	16

* *Expression 12 in 20-case has an un-paired parenthesis, and cannot be used in the experiment.*

This paper uses two sets of Boolean expressions from two different real systems (for details see Section 10.2) to observe the possibilities for their ILP constraint matrices to be diagonalized. One set has 20 expressions whose constraint matrices can be decomposed into different numbers of blocks varying from 1 to 43 with average 16, where the constraint matrices of two expressions in this set are both single identity matrix; another has 25 expressions whose constraint matrices can be decomposed into different numbers of blocks varying from 2 to 38 with average 9. This implies that significant size reduction of ILP problems of fault detections for Boolean expressions can be achieved as the original problem is bigger than the max block size.

In the following sections, the 20 expressions in [43] are referred to as 20-case and the 25 expressions as 25-case.

7.4 Determining the Difficulty of Sub-Problems

Each of the sub-problems is still an ILP. Experiments on the 20-case and 25-case showed that most of the sub-problems can be solved by an ILP solver such as GLPK or Lingo [49] quickly.

ILP problems in general are NP-hard, and thus often heuristics are used. Branch-and-Cut (B&C) [50] is one such method that involves running a branch-and-bound algorithm on the relaxed linear programming (LP) problem (by dropping the 0-1 constraints on variables), applies a practical linear program algorithms such as Simplex [31], and finally uses cutting planes to tighten the LP relaxations.

When an optimal solution is obtained and this solution has a non-integer value, $x'_{i'}$, for an integer variable, x_i, a cutting plane algorithm is used to add to the current LP relaxation two constraints, $x_i \leq \lfloor x'_{i'} \rfloor$ and $x_i \geq \lfloor x'_{i'} \rfloor$, such that resolving the relaxation will yield a different solution hopefully "less fractional".

When the relaxations of ILP have many fractional or non-integer values in an optimal solution, B&C tends to have many points to split, corresponding to more sub-problems to be solved, and the time to obtain the optimal solution increases dramatically. Thus, the number of non-integer values is an important indicator to the difficulties.

Experiments on the 20-case and 25-case showed if more non-integer values are obtained in the relaxation, the algorithm will take more time to complete. The second through the fifth columns in Table 1 shows the measurements of Boolean expressions in 20-case including the number of variables, the number of terms, the lengths (min-average-max) of terms, and the number (min-average-max) of occurrences of variables in an expression. The sixth has three portions to indicate the numbers of: non-trivial sub-problems/trivial sub-problems/identical sub-problems. For example, Expression 20-#1 has 2 non-trivial sub-problems; 10 trivial sub-problems, either identity matrix, or row matrix, or column matrix; and one identical sub-problem that has the same matrix structure with one of the non-trivial sub-problems and does not need to run ILP to solve it. The seventh is the number of fractional values when applying the relaxation of ILPs for cell-covering problems. The eighth column indicates whether to use an optimal approach or an approximate approach, and the last column is the time consumption in milliseconds. In terms of the number of fractional values and time to obtain the optimal results, it is observed in Table 1:

(1) No fractional values: Expressions 20-#1, 20-#2, 20-#4, 20-#6 through 20-#11, 20-#13, 20-#19 and 20-#20 have no fractional values, and time consumptions range from 16 to 313 milliseconds.
(2) Relatively large numbers of fractional values: Expression 20-#3 has up to 83 fractional values with 22 non-trivial sub-problems, one of which has up to 29 fractional values, and the rest in between, and it takes about 1,141 milliseconds, the longest one in 20-

TABLE 2
The Impacts of the Number of Fractional Values and the Sub-Problems on Time Comsuptions in 25-Case

Exp. IDs	# of variables	# of terms	Lengths of Terms	# of occurrences	# of sub-problems	# of fractional values	Optimal/ Approximate	Time (Milliseconds)
1	2	2	1 - 1.00 - 1	1 - 1.00 - 1	0 / 2 / 0	0	Optimal	0
2	12	10	3 - 3.00 - 3	1 - 2.50 - 10	1 / 11 / 1	52	Approximate	469
3	5	3	3 - 3.00 - 3	1 - 1.80 - 3	1 / 4 / 1	0	Optimal	15
4	5	3	3 - 3.00 - 3	1 - 1.80 - 3	4 / 3 / 1	0	Optimal	16
5	4	2	3 - 3.00 - 3	1 - 1.50 - 2	0 / 2 / 0	0	Optimal	0
6	8	4	5 - 5.50 - 6	2 - 2.75 - 4	4 / 2 / 3	0	Optimal	15
7	8	8	1 - 1.00 - 1	1 - 1.00 - 1	0 / 9 / 0	0	Optimal	16
8	6	3	4 - 4.00 - 4	1 - 2.00 - 3	2 / 4 / 1	0	Optimal	0
9	8	3	6 - 6.00 - 6	1 - 2.25 - 3	2 / 4 / 3	0	Optimal	16
10	9	5	2 - 2.40 - 3	1 - 1.33 - 3	5 / 0 / 1	5	Optimal	47
11	11	4	4 - 4.50 - 5	1 - 1.64 - 2	3 / 0 / 2	0	Optimal	47
12	5	5	1 - 1.00 - 1	1 - 1.00 - 1	0 / 6 / 0	0	Optimal	0
13	8	5	4 - 4.00 - 4	1 - 2.50 - 5	2 / 6 / 1	28	Optimal	15
14	11	4	3 - 4.50 - 6	1 - 1.64 - 2	4 / 0 / 1	0	Optimal	31
15	11	3	4 - 5.00 - 6	1 - 1.36 - 3	5 / 0 / 0	0	Optimal	62
16	12	5	3 - 3.60 - 4	1 - 1.50 - 3	6 / 0 / 0	5	Optimal	157
17	9	3	5 - 6.00 - 7	1 - 2.00 - 3	4 / 12 / 0	0	Optimal	31
18	5	2	4 - 4.00 - 4	1 - 1.60 - 2	2 / 5 / 0	0	Optimal	0
19	9	3	3 - 3.67 - 4	1 - 1.22 - 3	5 / 0 / 0	6	Optimal	31
20	14	10	5 - 5.80 - 7	1 - 4.14 - 8	5 / 0 / 6	0	Optimal	610
21	6	3	5 - 5.00 - 5	2 - 2.50 - 3	0 / 4 / 0	0	Optimal	0
22	14	9	7 - 7.33 - 8	3 - 4.71 - 9	15 / 0 / 2	0	Optimal	359
23	9	4	4 - 5.25 - 6	1 - 2.33 - 4	6 / 4 / 1	0	Optimal	31
24	13	4	10 - 10.50 -11	2 - 3.23 - 4	7 / 26 / 5	0	Optimal	78
25	16	6	7 - 7.50 - 8	2 - 2.81 - 6	12 / 0 / 1	25	Optimal	1,281

case; 20-#16 has 24 fractional values with 16 sub-problems, one of which has up to 16 fractional values, and the rest in between, and it takes less time than 20-#3, about 1,016 milliseconds, and more time than the rest of other expressions.

Expressions 20-#3 and 20-#16 have both 12 variables with more terms, 25 and 23, and relatively large number of occurrences of variables. Thus, in addition to the number of fractional values, the number of occurrences of variables and the number of Boolean variables may also impact the difficulties. Overall, expressions in 20-case take about one second to obtain their optimal solutions. However, the situations are different in 25-case as shown in Table 2 that has the same structure as Table 1 does.

(1) No or small number of fractional values: Expressions 25-#1, 25-#3 through 25-#9, 25-#11, 25-#12, 25-#14, 25-#15, 25-#17, 25-#18, and 25-#20 through 25-#24 have no fractional values, and time consumptions are all less than 100 milliseconds except for expressions 25-#20 and 25-#22 that have both 14 variables with 10 terms and 9 terms, relatively high lengths of terms and the occurrences of variables, spending 610 and 359 milliseconds, respectively. 25-#10, 25-#16 and 25-#19 have small numbers of fractional values, 5 or 6, and spending 31-141 milliseconds. 25-#13 with 8 variables has 2 non-trivial sub-problems with both 14 fractional values, and it has relatively small number of variables. 25-#25 has up to 16 variables and relatively small number of fractional values for non-trivial sub-problems, spending about 1.28 seconds.

Large number of fractional values: 25-#2 sees a different situation, with 12 variables, 10 terms and up to 52 fractional values for one single non-trivial sub-problem, and costs about 14 minutes with GLPK as the ILP solver to generate an optimal result, much higher than other expressions. The large number of fractional values is conspicuous, although the numbers of variables and terms are relatively large but not the largest ones. This suggests that the number of fractional values may be a better indicator than the number of variables to indicate the difficulty.

8 Approximate Approach

8.1 Cell-Covering Problem and Set-Covering Problem

The minimum cell-covering problem is to select a subset of cells from a set of cells to guarantee the detections of the nine types of faults in the fault hierarchy in Fig. 1. The minimum cell-covering problem is similar to the minimum set-covering problem [47], and approximate set-covering algorithms for the set-covering problem can be used to address the cell-covering problem by a transformation function.

Given a Boolean expression S, the topological structure of S and LIFs, LOFs and LRFs can be determined, i.e., the set of all possible faults U can be determined. Furthermore, the cells $X = \{x_1, \ldots, x_N, \mathrm{N} \leq 2n\}$ that can detect all the faults in U can be determined. Let H be a transformation function, $H(x_i)$ is a set of faults that can be detected by x_i, $x_i \in X$; and $H(X)$ is a set of $H(x_i)$ whose union equals the set of all possible faults U.

The minimum set-covering problem $\langle U, H(X) = \{H(x_1), \ldots, H(x_i), \ldots, H(x_N)\}\rangle$ identifies the smallest subset of $H(X)$. Assume the smallest subset of $H(X)$ is $H(X') = H(x_{1'}), H(x_{2'}), \ldots H(x_{i'}), \ldots, H(x_{N'}), N' \leq N$. The number of elements in $H(X')$, $|H(X')|$, depends on the selection of

$X' = \{x_{1'}, \ldots, x_{i'}, \ldots, x_{N'}, N' \leq N\}$. This corresponds to the smallest set $X' \subseteq X$ of cells that can detect all the faults in U.

Example 11. Take Fig. 8 as an example to illustrate the set-covering problem representation for $S = a + \bar{b}cd$ regarding LIFs:

$U = \{ab + \bar{b}cd, a\bar{b} + \bar{b}cd,\ ac + \bar{b}cd,\ a\bar{c} + \bar{b}cd,\ ad + \bar{b}cd,\ a\bar{d} + \bar{b}cd,\ a + a\bar{b}cd\}$, the set of all possible LIFs for expression $S = a + \bar{b}cd$ according to earlier analysis.

$H(X) = \{H(x_3), \ldots, H(x_{16})\} = \{\{a + a\bar{b}cd\}, \{a\bar{b} + \bar{b}cd, ac + \bar{b}cd,\ ad + \bar{b}cd\}, \{a\bar{b} + \bar{b}cd,\ ac + \bar{b}cd, a\bar{d} + \bar{b}cd\}, \{a\bar{b} + \bar{b}cd, a\bar{c} + \bar{b}cd,\ a\bar{d} + \bar{b}cd\},\ \{a\bar{b} + \bar{b}cd, a\bar{c} + \bar{b}cd, ad + \bar{b}cd\},\ \{ab + \bar{b}cd, ac + \bar{b}cd, ad + \bar{b}cd\},\ \{ab + \bar{b}cd, ac + \bar{b}cd, a\bar{d} + \bar{b}cd\},\ \{ab + \bar{b}cd, a\bar{c} + \bar{b}cd, ad + \bar{b}cd\}\}$.

Where each of the elements in $H(X)$ is a sub-set of set U regarding LIFs, and can be detected by a test case in X.

$X = \{x_3, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{16}\}$

The solution to the minimum set-covering problem $\langle U, H(X) = \{H(x_1), \ldots, H(x_i), \ldots, H(x_N)\}\rangle$ is a solution for the minimum cell-covering problem to find a minimum number of test cases to cover all the faults in U. The minimum set-covering problem is NP-complete [7], and quality approximate algorithms are available [9].

The experiments showed that, in 20-case, the time consumptions of the optimal results range from less than 1 to 1140 milliseconds with average 173. However, when applying the optimal approach to 25-case, all the expressions except for 25-#2 take time from less than 1 to 1,250 milliseconds with average 112 milliseconds, while 25-#2 take up to about 14 minutes (If using the commercial tool Lingo [49] as the ILP solver, the time is about 4.6 minutes. Our tool integrates the GLPK ILP solver [51]). In this situation, approximate algorithms are indeed good choices to solve the cell-covering problem.

8.2 Greedy Algorithm and Its Applications

The greedy algorithm used is an efficient approximation algorithm for the set-covering problem [9]. At each stage, the algorithm chooses the set that contains the largest number of uncovered elements. When applying this algorithm to the cell-covering problems, the algorithm can be described as follows:

Input: the set of all the faults U and a collection $H(X)$ of $H(x_i)$, $x_i \in X$, $X = \{x_1, \ldots, x_N, \mathrm{N} \leq 2^n\}$, where $H(x_i)$ is a subset of faults in U, and can be detected by cell x_i; and the union of the subsets $H(x_i)$ is U;

Output: set C containing a set of cells that can detect the faults in A.

(1) $A \leftarrow \emptyset$, $C \leftarrow \emptyset$.

(2) While $|A| < |U|$ do

Select a cell x_i, $x_i \in X$, to maximize $|A \cup H(x_i)|$, where $H(x_i) \in H(X)$;

$C \leftarrow C \cup x_i$, $A \leftarrow A \cup H(x_i)$, $H(x_j) \leftarrow H(x_j)/H(x_i)$ where $H(x_j) \in H(X)$ and $H(x_j) \neq H(x_i)$;

(3) Output A and C.

This algorithm is proved to produce a polynomial-time $(1 + \ln\gamma)$-approximation [9] for the minimum set-covering problem, where γ is the maximum cardinality of a subset in the input collection $H(X)$. When γ is equal to 1, each subset in $H(X)$ has only one element, i.e., each x_i can detect one fault only. In this case, the problem is a trivial one, and the union of those x_i is the optimal solution. When γ gets large, the probability of element overlapping among the subsets gets large, i.e., a fault might be detected by more than one test case x_i, and then the greedy algorithm can be applied to solve the covering problem. The worst computational complexity of the greedy algorithm is $O(2^n \cdot n \cdot m)$ to detect the three faults, LIFs, LOFs and LRFs, where n is the number of Boolean variables in the expression and m is the number of the terms.

Example 12. Based on Example 11 in Section 8.1, the following illustrates to obtain the test cases for $S = a + \bar{b}cd$ with the greedy algorithm regarding LIFs:

(1) $A \leftarrow \emptyset$, $C \leftarrow \emptyset$.

(2) At this moment $|A| < |U|$, as x_9 through x_{16} have the same larger number of faults being able to detect, pick up the first one, x_9.

Update A, C and $H(X)$, $A = \{a\bar{b} + \bar{b}cd, ac + \bar{b}cd,\ ad + \bar{b}cd\}$, $C = \{x_9\}$, and $H(X) = \{\{a + a\bar{b}cd\}, \{a\bar{d} + \bar{b}cd\}, \{a\bar{c} + \bar{b}cd,\ a\bar{d} + \bar{b}cd\},\ \{a\bar{c} + \bar{b}cd\},\ \{ab + \bar{b}cd\},\ \{ab + \bar{b}cd,\ a\bar{d} + \bar{b}cd\}, \{ab + \bar{b}cd, a\bar{c} + \bar{b}cd\}\}$. Now, x_{11}, x_{14}, and x_{16} have the same larger number of faults being able to detect, pick up the first one, x_{11}.

Update A, C and $H(X)$, $A = \{a\bar{b} + \bar{b}cd, ac + \bar{b}cd,\ ad + \bar{b}cd, \bar{c} + \bar{b}cd, a\bar{d} + \bar{b}cd\}$, $C = \{x_9, x_{11}\}$, and $H(X) = \{\{a + a\bar{b}cd\}, \{ab + \bar{b}cd\}, \{ab + \bar{b}cd\}, \{ab + \bar{b}cd\}\}$. This time, x_3, x_{13}, x_{14}, and x_{16} have the same number of faults being able to detect, pick up the first one, x_3.

Update A, C and $H(X)$, $A = \{a\bar{b} + \bar{b}cd, ac + \bar{b}cd,\ ad + \bar{b}cd, a\bar{c} + \bar{b}cd, a\bar{d} + \bar{b}cd, a + a\bar{b}cd\}$, $C = \{x_9, x_{11}, x_3\}$, and $H(X) = \{\{ab + \bar{b}cd\}, \{ab + \bar{b}cd\}, \{ab + \bar{b}cd\}\}$. Currently, x_{13}, x_{14}, and x_{16} have the same number of faults being able to detect, pick up the first one, x_{13}.

Update A, C and $H(X)$, $A = \{a\bar{b} + \bar{b}cd, ac + \bar{b}cd,\ ad + \bar{b}cd, a\bar{c} + \bar{b}cd, a\bar{d} + \bar{b}cd, a + \overline{aa}\bar{b}cd, ab + \bar{b}cd\}$, $C = \{x_9, x_{11}, x_3, x_{13}\}$, and $H(X) = \{\emptyset\}$. At this moment, A is equal to U with seven LIFs, meeting the stopping criteria and C is the solution obtained.

(3) Output A and C.

The optimal result that has three cells but C has four cells regarding LIFs. Three optimal solutions are available $\{x_3, x_{10}, x_{16}\}$, $\{x_3, x_{11}, x_{13}\}$, and $\{x_3, x_{12}, x_{14}\}$ as shown in Fig. 8.

8.3 Elaborating the Approximation Ratio

The greedy algorithm achieves good results with the following conditions:

(1) The approximation ratio, $(1 + \ln\gamma)$ [9], of the greedy algorithm approaching to 1 implies that γ is approaching to 1, and the approximation solution is close to the optimal solution. For example, γ will be 1 if the coefficient matrix is an identity matrix, and this means one cell detects one fault, and one fault can be detected by one cell. In this case, $H(x_i)$ is a bijective function or a one-to-one function.

(2) Similar to (1), there is another case when γ is equal to 1, i.e., each $H(x_i)$ has only one fault that can be detected a subset of cells, C_i, and for any different i and j, $C_i \cap C_j = \phi$. In this case, $H(x_i)$ is a multi-

value function, and the corresponding coefficient matrix is a row matrix as discussed in Section 7.3.

(3) For any different x_i and x_j, if $H(x_i) \cap H(x_j) = \phi$, then the greedy algorithm can obtain the optimal solution no matter how large the value of γ is. In this case, each sub-problem of ILP is a trivial one with the constraint coefficient matrix as a single column as described in Section 7.3.

With regard to the observations of 20-case and 25-case, when γ is less than or equal to 3, the greedy algorithm can obtain the optimal solutions. However, when γ is equal to or greater than 4, the greedy algorithm may not obtain the optimal solutions.

8.4 Solving Cell-Covering Problems with Greedy Algorithm

When a cell-covering problem is modeled as an ILP, the constraint coefficient matrix can be interpreted as follows: each row of the matrix corresponds to a fault in U, and each fault in U has a row in the matrix; each column of the matrix corresponds to a test case x_i in X; the set of 1's on each column corresponds to a transformation function $H(x_i)$ in $H(X)$, i.e., a subset of faults that can be detected by test case x_i on ith column. Then the greedy algorithm can be described as follows:

Input: a constraint coefficient matrix of ILP for a cell-covering problem with p rows and q columns;
Output: C containing a set of cells that can detect the faults in covered rows r.
(1) $r \leftarrow 0$, $C \leftarrow \emptyset$.
(2) While $r < p$ do
Select a column that corresponds to a cell x_i, $x_i \in X$, to maximize ($r +$ the number of 1's in that column);
$C \leftarrow C \cup x_i$, $r \leftarrow$ ($r +$ the number of 1's in the column), and mark the corresponding rows as covered by flipping 1 to 0.
(3) Output r and C.

The worst computation complexity of the above algorithm is O($p \cdot q$). In the algorithm, the maximum value of the columns, q, is 2^n, and n is the number of variables of a Boolean expression. For LIFs, the maximum value of the rows, p, is $m \times (n - \alpha_i) \times 2$, and m is the number of terms and α_i is the number of literals in the term α_i; for LOFs, the maximum value of the rows, p, is $m \times \alpha_i$; and for LRFs, the maximum value of the rows, p, is $m \times \alpha_i \times (n - \alpha_i) \times 2$.

By diagonalizing the constraint coefficient matrix as described in Section 7.3, the original cell-covering problems of ILPs in both 20-case and 25-case are decomposed into either trivial sub-problems, or non-trivial sub-problems, or identical sub-problems as shown in Tables 1 and 2 at the sixth columns. The trivial sub-problems are easy to obtain the optimal solutions. All the non-trivial sub-problems have smaller sizes than the original ones, and if the non-trivial sub-problems are decided to be solved using the greedy algorithm, the number of rows and columns of the sub-problems, p' and q', are smaller than p and q, and thus less time to solve the individual sub-problems.

It is more important to notice that the maximum cardinality, $\boldsymbol{\gamma}'$, for each the non-trivial sub-problems may be reduced after the diagonalization compared with the original one, γ. For example, the maximum cardinality of the original cell-covering problem for 25-#2 in 25-case is 90, while the maximum cardinality of the non-trivial sub-problem is reduced to 25. This implies the approximate ratio of the greedy algorithm is reduced, and the greedy algorithm has a tighter bound for the sub-problem, i.e., with a higher confidence on the approximate result.

9 A Hybrid Approach

9.1 Process of the Hybrid Approach

This section proposes a hybrid approach to solve the cell-covering problem with six steps as shown in Fig. 10:

(1) Pre-process Boolean expressions: Given a Boolean expression S in DNF (or in CNF, and the faults are defined in terms of CNF as well), get the number n of variables, the number m of terms, the set of unique true cells of term T_i ($i = 1, \ldots, m$), and the set of STCs of S. The computational complexity is $O(2^n \cdot m)$, i.e., for each cell in K-map, every term of S is scanned to see if it is true regarding the cell, if yes, its counter increases one; finally if the counter is equal to 1, then the cell goes to the set of UTCs of the term, if the counter is greater than 1, then the cell is put into the set of STCs of S.

(2) Generate constraint coefficient matrix and objective coefficient vector: Given fault type(s) $f \in \mathcal{F}$, generate constraint coefficient matrix and objective coefficient vector for the minimum cell-covering problem. The algorithms of constraint coefficient matrix generation have been described in Section 6.2. When generating a constraint coefficient matrix to detect LIFs, LOFs and LRFs, the computational complexity is $O(m \cdot n)$, i.e., for each of faults, consider every dimension of each term of the expression in K-map, where the dimension varies from 1 to $\lceil \frac{n}{2} \rceil$.

(3) Divide the original problem into sub-problems: Diagonalize the coefficient matrix by performing columns' interchanges and primary transformations of matrix as described in Section 7.3. Divide the objective function into sub-objective functions. The computational complexity is $O(n^4)$. The sub-matrices on the diagonal of the transformed coefficient matrix and their corresponding objective functions correspond to the sub-problems.

(4) Conquer the sub-problems:
a) Simplify the sub-matrices by removing its 0-value rows and sort its rows by the 0-1 numbers in rows and categorize the sub-problems, such that the sub-problems with the same matrix are in the same category. Solve one sub-problem in every category.
b) The sub-problems of a category are *trivial* if they turn out to be the following situations:
- The sub-problem's matrix is an identity matrix. Then vector <1, 1 ... 1> is the sole optimal solution for such sub-problems.
- The sub-problem's matrix has only one row. Any identity vector, i.e., with one element as 1 and the rest as 0, is an optimal solution. Take vector <1, 0... 0> as the

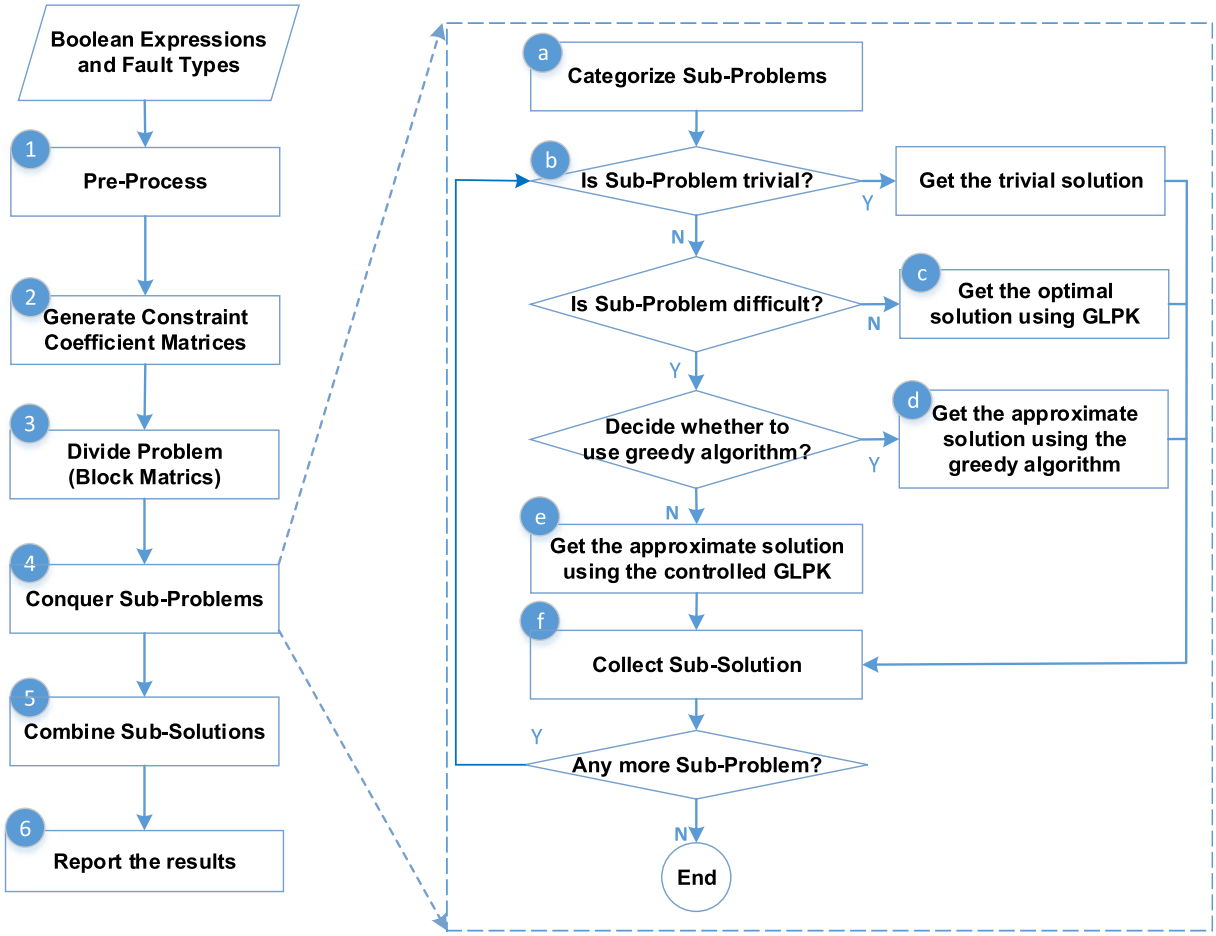


Fig. 10. The process to generate test cases for Boolean expressions.

optimal solution for the trivial sub-problem. In this case, the fault that the constraint row aims to detect can be identified by any solution of those vectors.

- The sub-problem's matrix has only one column. The identity vector with one element is the optimal solution. Each constraint row in the column targets one fault. That is the optimal solution can identify all those faults.

c) A sub-problem is considered as difficult if the optimal solution for its relaxation problem has many none-integer values. The threshold can be determined by experiments and is discussed in Section 7.4. Otherwise, the optimal solution is obtained by an ILP solver such as GLPK.

d) A "difficult" sub-problem is considered to be suitable to be solved by the greedy algorithm if the approximation ratio is less than a threshold that can be determined by experiments, and this is shown in Section 8.3. In addition, if time priority is high, use the greedy algorithm to obtain an approximate solution as described in Section 8.4.

e) Take the LP relaxation solution as the lower bound, calculate the upper bound of the integer solution using an ILP solver. The calculation will terminate when a "difficult" sub-problem is solved by an ILP solver guided by two conditions: (i) the branch & cut process is terminated once the gap between the lower bound and the upper bound is small enough; (ii) the branch & cut process is terminated if the time runs out. The thresholds of the gap and time are determined by experiments to be discussed in Section 9.2.

f) Save the solution of the sub-problem. If there is more sub-problem not processed yet, continue to b), otherwise to (5).

(5) Combine the sub-solutions: Get the final optimal solution by concatenating the sub-solutions of all the sub-problems.

(6) Report the results: Associate the ILP results with cell coordinates and obtain test cases; and report the number of test cases and the time cost.

The hybrid approach uses the number of non-integer values in the optimal solution of LP relaxation in the first round to measure the difficulties to solve the ILP. Based on the experiment results, the threshold is set up as 40. For example, Expression 25-#2 in 25-case has a sub-problem with 52 non-integer values in the optimal solution of the relaxation of ILP in the first round, obviously greater than 40, and considered as a difficult sub-problem.

9.2 Determining Gaps and Timeouts

The controlled ILP terminates the process of branch & cut when the gap between the current optimal integer solution and the lower bound is less than a threshold.

Experiments on 20-case and 25-case show that when the size of the optimal solution is not large, say less than 20, setting the gap within less than or equal to 5 can largely reduce the computing time, otherwise setting the gap as the floor of 10% of the optimal solution, denoted as $^{\triangle}Z$. In the hybrid process, the threshold of the gap is set as $\mathrm{Max}^{\triangle}\{Z, 5\}$.

To approach the threshold of the gap, the controlled ILP may still take a long time to get the results in some situations, e.g., 25-#2 in 25-case. To deal with such situations, the ILP is terminated when there is no update for the upper bound with a certain period of time, setting as 100 seconds in the experiment, and a feasible integer solution has been reached.

In general, the controlled ILP of a cell-covering problem does not target optimal solutions, and obtains a satisfied solution by a pre-defined threshold of the gap between the current optimal integer solution and the lower bound, or by time-outs to terminate the process.

10 Experiments

This section specifies the research questions first, then presents machine configurations and data sets, demonstrates the experiment results based on different metrics. The experiments evaluate the proposed techniques with respect to test sizes, computation time, and fault detection capabilities.

10.1 Research questions

MUMCUT approach has been proved to guarantee to detect the nine types of single faults in Fig. 1. Minimal-MUMCUT obtains smaller sizes of test cases while guarantees to detect the nine types of single faults and provided the empirical results. TRF-TIF is another approach based on mutation-based approach that is reported to obtain smaller sizes of test cases effectively and guarantees to detect the nine types of single faults. The three researches all provided their empirical results with the 20 expressions in Appendix A2, available in the online supplemental material. Cell-covering approach is theoretically proved to guarantee to detect the nine types of single faults. Therefore, the experiment first tries to answer the following two questions:

(1) Can cell-covering approach obtain the minimal sizes of test cases compared with minimal-MUMCUT and TRF-TIF approach?
(2) Can the different optimizations of ILP to cell-covering approach achieve better results compared with minimal-MUMCUT and TRF-TIF approach in terms of performances?

Kaminski and Ammann applied optimization to logic testing, developed an ILP model based on minimal-MUMCUT strategy, and reported their optimal average result for the 20 expressions. The experiment in this paper tries to answer one more research question:

(3) Can cell-covering approach and the optimal Minimal-MUMCUT guarantee to detect fault variants other than the single-faults, such as, double-faults in the fault hierarchy?

The three research questions tie back to the contributions stated in the introduction. The first one is about evaluating the fault modeling and detection and mainly related to contributions (1) through (4); the second one is about evaluating the ILP optimization and hybrid approaches, and mainly related to contributions (5) through (8); the third is about evaluating the capability of double-fault detection. Empirical experiments regarding the three questions are performed using the developed tool, related to the contribution (9); and all the activities in this section is related to the contribution (10).

To be specific, Section 10.2 presents the experiment settings; Section 10.3 compares the test sizes produced by three approaches mentioned in the first question; Section 10.4 compares the times to generate the test cases; Section 10.5 evaluates the approaches with randomly generated data sets; Section 10.6 compares the fault detection capabilities, especially double faults.

10.2 Experiment Settings

The experiments are done on a PC with Intel Core i3-2310 Duo CPU-2.10 GHz, memory 4 GB, and 64-bit Windows 8 Operating System. A tool has been developed and is available at Github [51]. The experiments uses two data sets to evaluate the related approaches: 1) 20 Boolean expressions extracted from TCAS II, an aircraft collision avoidance system, by Weyuker, Goradia, and Singh [43] as shown in Appendix A2, available in the online supplemental material, among them expression #12 has an unmatched parenthesis, and thus removed from the experiments; and 2) 25 Boolean expressions collected from a marine navigation system with three sub-systems by the team-members of this paper as shown in Appendix A3, available in the online supplemental material. The two systems represent examples of formally specified real-world process-control systems, which are safety-critical and need high quality requirements.

To further observe the characteristics, 4 sets of Boolean expressions, totally 88 Boolean expressions, are randomly generated based on the constructions of the 20 expressions and 25 expressions: the number of Boolean expressions in each set varies from 20-25, the number of terms in each Boolean expression varies from 2-17, the number of literals in a term varies from 1-7, and the number of variables in a Boolean expression varies from 2-16. In the following sections, the 88 expressions is referred to as 88-case.[2]

10.3 Comparing Test Sizes

This section compares the test sizes of cell-covering with that of minimal-MUMCUT, and TRF-TIF mutation testing [21]. Minimal-MUMCUT and TRF-TIF are chosen because they are leading mechanisms.

10.3.1 Comparison with Minimal-MUMCUT

Fig. 11 compares for the 20-case the test sizes to detect LIFs, LOFs, and LRFs using cell-covering and minimal-MUMCUT. Two sets of data applying minimal-MUMCUT are used: one comes from [20], the other is from the website (http://cs.gmu.edu:8080/offutt/coverage/MinimalMUMCUTCoverage, the version dated as April 24, 2013). Compared to minimal-MUMCUT, cell-covering needs less

2. The 88 expressions are available on github: https://github.com/PKU-YU/Cell-Covering-Tool/blob/master/CellCoveringTool/expression/88_DNF_Generated.txt

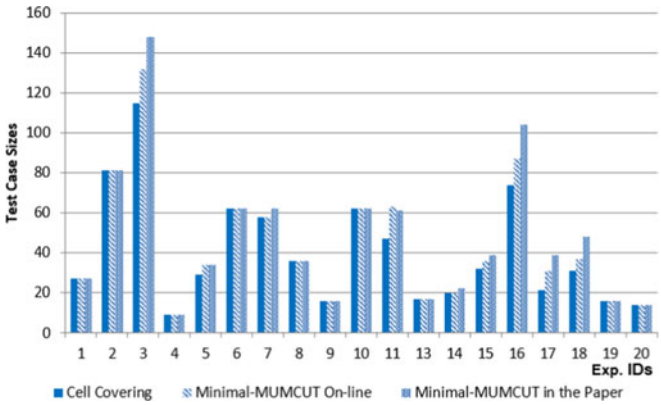


Fig. 11. Test sizes: Cell-covering versus minimal-MUMCUT (20-case).

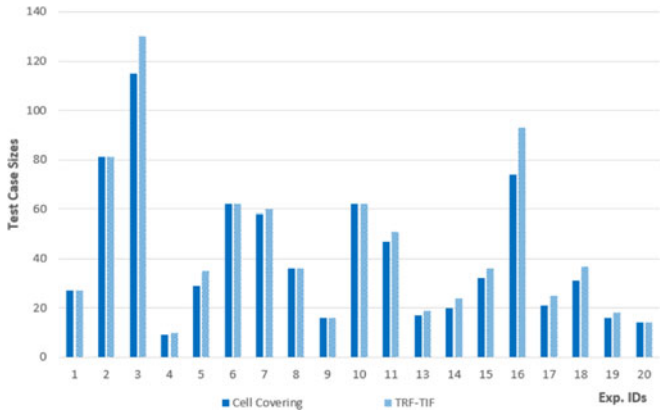


Fig. 13. Test sizes: Cell-covering and TRF-TIF (20-case).

(expressions 20-#3, 20-#5, 20-#11, 20-#15 through 20-#18) or at most equal number of test cases (expressions 20-#1, 20-#2, 20-#4, 20-#6, 20-#8 through 20-#10, 20-#13, 20-#19 and 20-#20 using minimal-MUMCUT for both the online and the report, and 20-#7 and 20-#14 only for the online).

For some expressions, minimal-MUMCUT can get the optimal results as cell-covering does. It is interesting to see the characteristics of those expressions that minimal-MUMCUT obtains optimal results.

1) Expressions have special structures. For example, each term in 20-#8 and 20-#9 has all Boolean variables appearing (in positive or in negative format), i.e., they have neither LIFs nor LRFs. In this case, to detect the faults in the hierarchy in Fig. 1, the test cases to detect TOFs and LOFs must be generated. In fact, each term in 20-#8 has 8 valid fault instances, also called valid mutants (i.e., not equivalent ones regarding the original expression) per LOFs, and one valid fault per TOF, and each fault instance has a unique test case to detect, i.e., 9 test cases for each term. 20-#8 has 4 terms, and has 36 in total that is the optimal result, and can be obtained by MUMCUT.
2) Expressions have small sizes of input space. For example, 20-#1 has 7 variables with five terms, among them, four terms with one variable not appearing, and one term with two variables not appearing. The four terms each have one unique UTP that must be selected, and the one term, meeting the feasibility of MUTP, has four UTPs and two are selected. It has 26 NFPs in total and 21 NFPs are selected based on P/CUTPNFP [20]. These expressions can obtain the optimal solution with MUMCUT approach. In addition, other expressions, such as expressions 20-#4, 20-#6 and 20-#10 have similar situations with relatively small sizes of input space, less than 100.

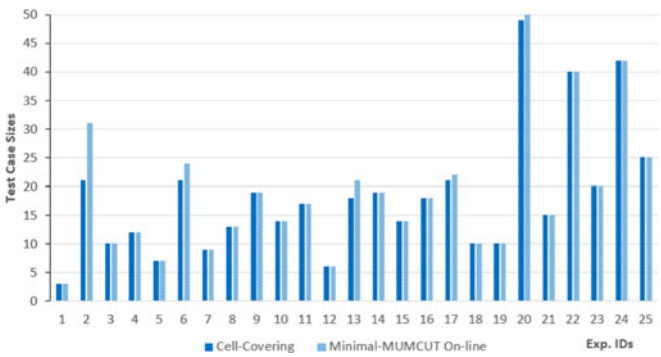


Fig. 12. Test sizes: Cell-covering and minimal-MUMCUT (25-Case).

However, not every expression with small sizes of input space can obtain optimal solutions with MUMCUT as cell-covering does, e.g., 25-#6 and 25-#13 in 25-case as shown in Fig. 12 have the sizes of input space, 67 and 99, respectively.

For 25-case, we used the online minimal-MUMCUT tool to calculate the test sizes. Fig. 12 shows cell-covering dominates minimal-MUMCUT as it needs less or equal number of test cases. Among them, cell-covering needs less test cases than that of minimal-MUMCUT for expressions 25-#2, 25-#6, 25-#13, 25-#17, and 25-#20, and the rest of expressions have the same test sizes regarding the two approaches.

Expressions 25-#1, 25-#3, 25-#4, 25-#5, 25-#8, 25-#9, 25-#15, 25-#18, and 25-#21 have one or two variables without appearing in the terms, and those kinds of expressions have relatively small total test sizes to choose from. Although expressions 25-#10, 25-#11, 25-#14, 25-#15, 25-#19, 25-#22 through 25-#25 have relatively large test sizes, and these expressions have special structures or properties, e.g., expressions 25-#19, 25-#22, 25-#24, and 25-#25 meet the feasibility of MUTP, and the rest meet the feasibility of P/CUTPNFP [20].

Minimal-MUMCUT can obtained optimal solutions when expressions have special structures, such as 20-#8 and 20-#9 in 20-case; or have relatively small sizes of test case space to select from, but cannot guarantee to obtain optimal solutions.

10.3.2 Comparison with TRF-TIF

Fig. 13 compares for the 20-case the test sizes to detect LIFs, LOFs, and LRFs using cell-covering and TRF-TIF [21]. Cell-covering needs less (expressions 20-#3 through 20-#5, 20-#7, 20-#11, and 20-#13 through 20-#19) or at most equal number of test cases (expressions 20-#1, 20-#2, 20-#6, 20-#8 through 20-#10, and 20-#20).

Fig. 14 compares for the 25-case the test sizes to detect LIFs, LOFs, and LRFs using cell-covering and TRF-TIF. Cell-covering needs less (expressions 25-#2, 25-#6, 25-#13, 25-#16, 25-#17, 25-#19, 25-#20, 25-#22, and 25-#25) or at most

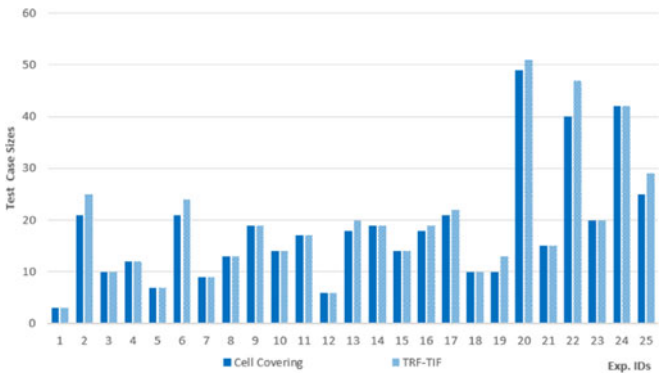


Fig. 14. Test sizes: Cell-covering and TRF-TIF (25-Case).

equal number of test cases (expressions 25-#1, 25-#3 through 25-#5, 25-#7 through 25-#12, 25-#14, 25-#15, 25-#18, 25-#21, 25-#23, and 25-#24).

As stated in Appendix D, available in the online supplemental material, TRF-TIF uses a greedy algorithm, and can generate optimal solutions under certain conditions as stated in Section 8.3. It is interesting to elaborate the related observations in the experiments in 20-case and 25-case.

When γ is one, the greedy algorithm can obtain an optimal solution. 20-#8 and 20-#9 in 20-case meet this condition and get the optimal result. In fact, these two expressions have all literals appearing in each term, have no faults of LIF and LRF, and only need to generate test cases to detect LOFs and TOF. 20-#8 has only 36 possible changes of topological structure, and each change corresponds to one and only one cell, and the cell is the test case to detect the corresponding fault. Similarly, 20-#9 has only 16 possible changes of topological structure, and each change corresponds to one and only one cell, and the cell is the test case to detect the corresponding fault.

When a test case can detect more than one mutant, but no mutants can be detected by more than one test case, then the greedy algorithm can obtain the optimal solution no matter how large the value of γ is. 25-#7 and 25-#12 in 25-case meet this condition and get the optimal result with the values of γ as 56 and 20, respectively. In fact, these two expressions have only one literal appearing in each term, have no faults of LOFs, and one only needs to generate test cases to detect LIFs, LRFs and TOFs. 25-#7 has only 176 possible changes of topological structure, and only 9 cells are needed to detect

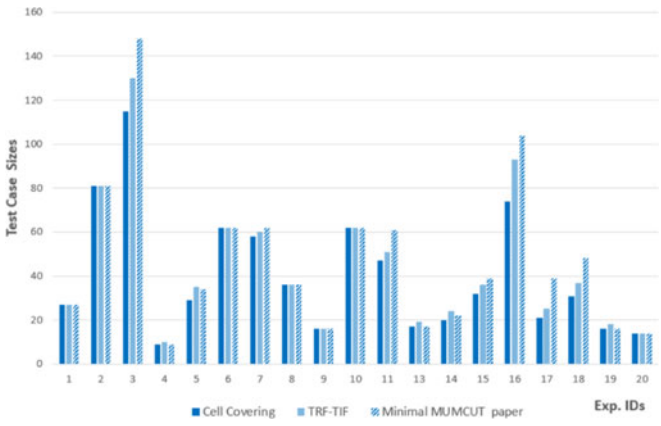


Fig. 15. Test sizes: Cell-covering minimal-MUMCUT and TRF-TIF (20-case).

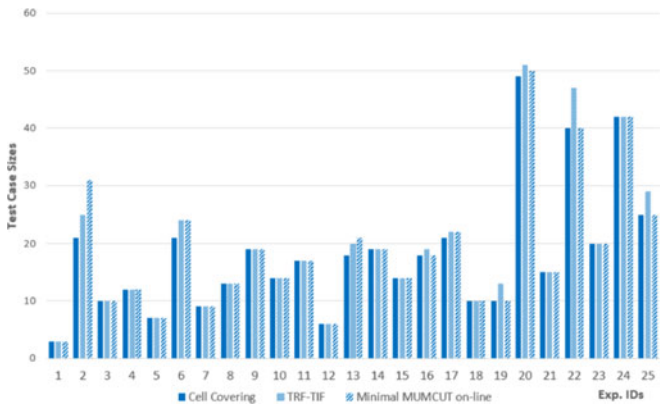


Fig. 16. Test sizes: Cell-covering minimal-MUMCUT and TRF-TIF (25-Case).

these faults. Among them, the largest fault set has 56 faults (i.e., $\gamma = 56$) all of that can be detected by one cell and the cell can only detect the faults in this set. The other fault sets of 25-#7 have smaller sizes than the largest one but with the similar properties. Similarly, 25-#12 in 25-case has the alike property, whose largest fault set has 20 faults (i.e., $\gamma = 20$) all of that can be detected by one cell and the cell can only detect the faults in this set.

As such expressions as 20-#8 and 20-#9, 25-#7 and 25-#12 have and only have one set of test case result that can guarantee to detect the nine types of single faults. In fact, in such a situation any approach that can get a set of test cases guaranteed to detect the nine types of single faults can obtain the optimal result, such as minimal-MUMCUT and TRF-TIF approach.

10.3.3 Comparisons Among Three Approaches

Fig. 15 shows the comparison of test sizes among the three approaches in the 20-case. TRF-TIF and minimal-MUMCUT have no dominance relation, i.e., sometimes TRF-TIF obtains better results than minimal-MUMCUT, but sometime the reverse turns out. In 20-case, cell-covering needs the least number of test cases for expressions 20-#3, 20-#5, 20-#7, 20-#11, and 20-#14 through 20-#19, or equal number of test cases for expressions 20-#1, 20-#2, 20-#8 through 20-#10, and 20-#20, and less number of test cases than either TRF-TIF or minimal-MUMCUT for the rest of expressions.

Again, TRF-TIF and minimal-MUMCUT have no dominance relation in 25-case as shown in Fig. 16. Cell-covering needs the least number of test cases for expressions 25-#2, 25-#6, 25-#13, 25-#17, and 25-#20, equal number of test cases for expressions 25-#1, 25-#3 through 25-#5, 25-#7 through 25-#12, 25-#14, 25-#15, 25-#18, 25-#21, 25-#23 and 25-#24, and less number of test cases than either TRF-TIF or minimal-MUMCUT for the rest of expressions.

10.4 Comparing Times to Generate Test Cases

10.4.1 Comparing Approaches with the 20-case

Fig. 17 compares time consumptions for generating test cases of the 20-case that guarantee to detect the faults in the fault hierarchy in Fig. 1 between cell-covering with optimal approach and minimal-MUMCUT. Note that Minimal-MUMCUT runs on an on-line server (http://cs.gmu.

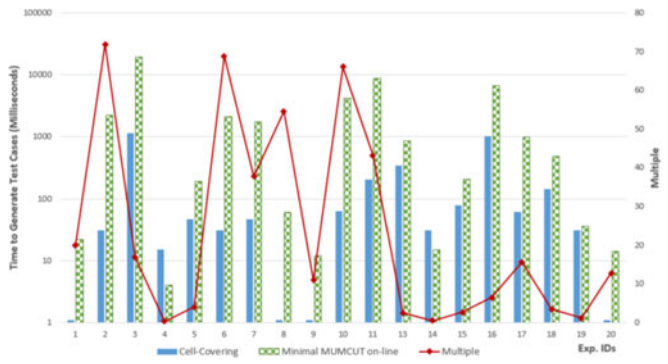


Fig. 17. Time consumption: Cell-covering and minimal-MUMCUT (20-case).

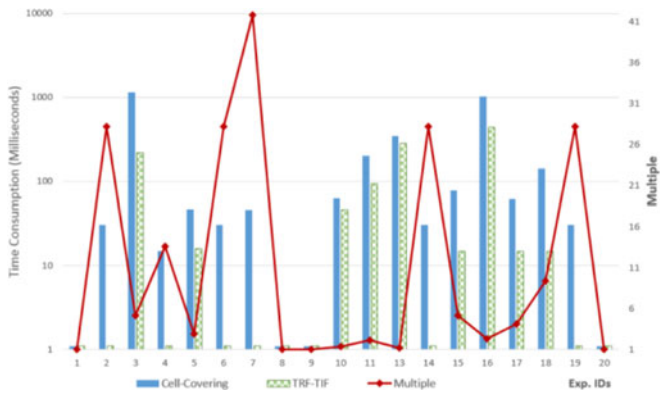


Fig. 18. Time consumption: Cell-covering and TRF-TIF (20-case).

edu:8080/offutt/coverage/MinimalMUMCUTCoverage). It would be ideal that all the programs of the algorithms to be compared are run on a same machine with the exact same environment. To respect the original effect of Minimal-MUMCUT, the comparisons of test sizes with Minimal-MUMCUT are using the paper results [20] and the on-line results. Correspondingly, the comparisons of time consumptions with minimal-MUMCUT are also using the ones output by the on-line tool.[3] Although this would make the time comparisons not that accurate, the trend should be able to be observed.

The times of cell-covering in Fig. 17 are the average values of 10 runs; the vertical axis on the left hand is logarithmically scaled with base 10, and the vertical axis on the right hand is the multiple of the time of minimal-MUMCUT vs. the time of cell-covering.

The time ratios of minimal-MUMCUT vs. cell-covering on the right-hand of the vertical axis indicate minimal-MUMCUT takes more time than the cell-covering with the ratios varying from 2.5 up to 72, except that cell-covering takes 2-3 times than minimal-MUMCUT for 20-#4 and 20-#14.

The time consumption of expressions 20-#1, 20-#8, 20-#9, and 20-#20 is less than one millisecond, and the machine reports 0 millisecond. As zero has no logarithm and the logarithm of less than 1 is negative, the vertical axis of Excel 2013 starts from 1, and the time with 0 is indicated as 1 instead in Figs. 14, 15, 16, and 17.

Compared with cell-covering using an optimal approach, TRF-TIF achieves a good performance in terms of time consumption as shown in Fig. 18. TRF-TIF takes almost the same time to generate test cases for expressions 20-#1, 20-#8, 20-#9 and 20-#20, and less time for the rest of expressions, especially for 20-#6 and 20-#7. The test case generation algorithm for TRF-TIF is actually applying a greedy approximation algorithm that can obtain a solution in a reasonable time.

10.4.2 Comparing Approaches with the 25-Case

In 20-case, overall, cell-covering spends less time than that of minimal-MUMCUT, and more time than that of TRF-TIF. However, in 25-case, 25-#2 spends relatively long time to generate the optimal solution, up to 881,813 milliseconds (about 14.7 minutes) with GLPK solver.

25-#2 has 12 variables and 10 terms, and each term has 3 variables with a large volume of NFPs. Selecting a small number of optimal solutions from such a large volume of NFPs incurs a relatively long time.

One can use the hybrid approach for cell-covering problem described in Section 9 to reduce the time for generating test cases while maintaining the optimality for most expressions. The experiment in Fig. 19 shows that TRF-TIF takes the least time to generate test cases for all the expressions in 25-case, and minimal-MUMCUT takes more time than the proposed hybrid approach (indicated as CC-Hybrid in Fig. 19) for most expressions.

The hybrid approach reduces the time to generate test cases for 25-#2 to 693 milliseconds and increases the test sizes by 4 up to 25. The time reduction rate is $881{,}813/693 = 1{,}272.46$, and the test size growth rate is $25/21 = 1.19$.

The experiment results in Fig. 20 do not include 25-#2, but the rest of expressions in 25-case. The figure shows the comparisons of test case sizes and the time consumptions of test case generation between the hybrid approach and the optimal approach for cell-covering problem solving, indicated as CC-Hybrid and CC-Optimal, respectively. It is observed that the test sizes of the hybrid approach for the other expressions keep the same, and the time consumptions almost keep the same, except that the hybrid approach takes more time for 25-#3 and cell-covering takes more time for 25-#18, but in both cases, the time consumptions are not more than 15 milliseconds.

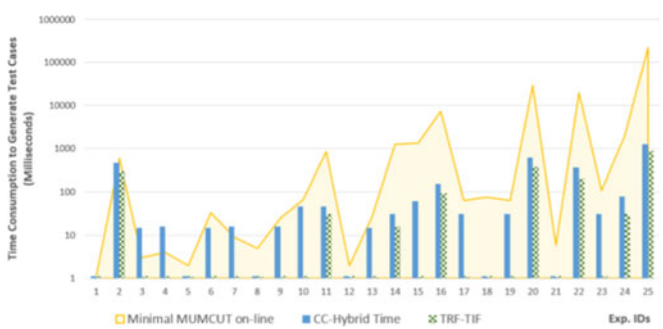


Fig. 19. Time consumption: Cell-covering, minimal-MUMCUT and TRF-TIF (25-Case).

3. It would be ideal to have the comparing algorithms run with the same machines configurations. As minimal-MUMCUT only provided on-line tool, both the test sizes and time consumptions are all collected based on the on-line results.

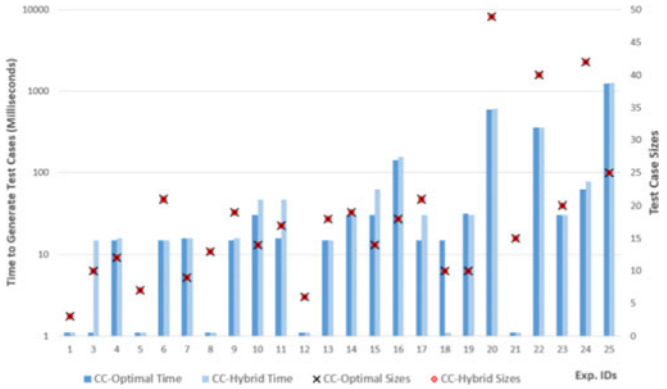


Fig. 20. Time consumption changes with no test case sizes changes. (25-Case).

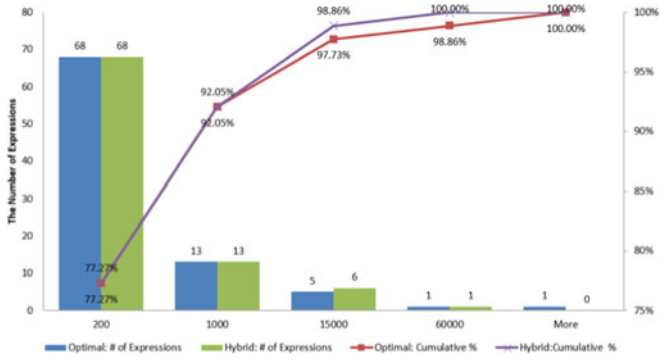


Fig. 21. Performances: Pure optimal approach versus hybrid approach.

10.5 Validating with Generated Data Sets

To further evaluate the hybrid approach, four more sets with a total of 88 expressions are generated based on the distributions of the number of Boolean expressions in one set, the number of variables and terms in a Boolean expression, the number of literals in a term as 20-case and 25-case. The idea is that these test cases will have similar characteristics of Boolean expressions used in real software.

In the four created sets, the numbers of expressions vary from 20 to 25, the numbers of variables range from 2 to 16, the numbers of terms in a Boolean expression vary from 2 to 17, and the numbers of literals in a term fluctuate from 1 to 7.

Most of expressions (68 or about 77.27 percent) spend the time ranging from less one millisecond to 200 milliseconds to get optimal results; 13 expressions spend the time ranging from 200 milliseconds to one second. Thus, 92.05 percent of expressions can obtain optimal results within one second. However, 88-#42 in 88-case takes about 15 minutes (5 minutes using Lingo as the ILP solver) to obtain optimal results. It has up to 60 fractional values in the relaxation of the ILP with 16 variables and the lengths of the terms range from 2 to 6. But the hybrid approach took less than 6 seconds to obtain results, and increase test cases by 2 only (from 28 to 30).

Fig. 21 compares the performance of the optimal approach versus the hybrid approach, where the horizontal axis is five bins of time consumptions (in the unit of millisecond), i.e., [0, 200], (200, 1,000], (1,000, 4,000], (4,000, 60,000], greater than 60,000; the vertical axis on the left hand is the number of expressions that fall into a bin; and the vertical axis on the right hand is the cumulative percentages of the numbers of expressions that fall into bins.

There is no difference in the numbers of expressions in the first two bins, i.e., time consumption is not more than 4 seconds. 88-#42 falls into the fifth bin with the optimal approach, and falls into the fourth bin with the hybrid approach. That is all the 88 expressions with the hybrid approach can obtain either optimal or near-optimal results within one minute.

In this experiment, each of the executions is performed 15 times, and the averages and standard deviations are considered to evaluate the stability of these approaches. Fig. 22 shows as an example of those values of maximum, minimum, averages and standard deviations for the expressions in the first bin in Fig. 21. It is observed that all the expressions in the bin except for the expressions with zero millisecond have certain degrees with variations. The expressions in the first bin are solved with ILP solver in both the optimal solution and the hybrid approach on a same machine, and the fluctuation came from the randomization.

All the expressions in 20-case can use the optimal solution, and only one expression in 25-case needs to use the approximate solution. Only one expression out of 88 needs to use the approximate solution. In other words, in most cases, optimal solutions can be used.

10.6 Comparing Double Fault Detection Capability

Garrett and Ammann applied an optimization approach to minimizing a minimal-MUMCUT test set. The minimal-MUMCUT strategy [18] can be described as follows:

```
For each term X
Generate MUTP tests for X
  If the MUTP criterion is infeasible for X
  For each literal x in X
    Generate CUTPNFP tests for x
    If the CUTPNFP criterion is infeasible for x
    Generate MNFP tests for x
    End For
  Else
  Generate a NFP for each literal x in X
End For
```

Where *infeasible* means that one cannot obtain the test points meeting certain criterion, e.g., MUTP that selects multiple unique true points for each term such that all literals not in the term attain values 1 and 0. The approach generates IP constraints to meet MUTP to guarantee detecting all LIFs; if MUTP is infeasible for a term and CUTPNFP is feasible for all literals in that term, generates IP constraints to satisfy LRF detection for each literal; if both MUTP and CUTPNFP are infeasible, it will add a single constraint to detect the LOF for that literal.

Kaminski and Ammann used the same 20 Boolean expressions of TCAS II [43] to perform an empirical study and obtain the minimum test case sizes based on the optimization approach. They presented the average results of the test case sizes, 40.37 on page 335. Their tool is on Web site (http://cs.gmu.edu:8080/offutt/coverage/MinimalMUMCUTCoverage). To obtain the results, one needs to use a separate optimal solver. This paper uses Lingo to solve the optimal model predicate-by-predicate. The results show that the optimal model described in [19]

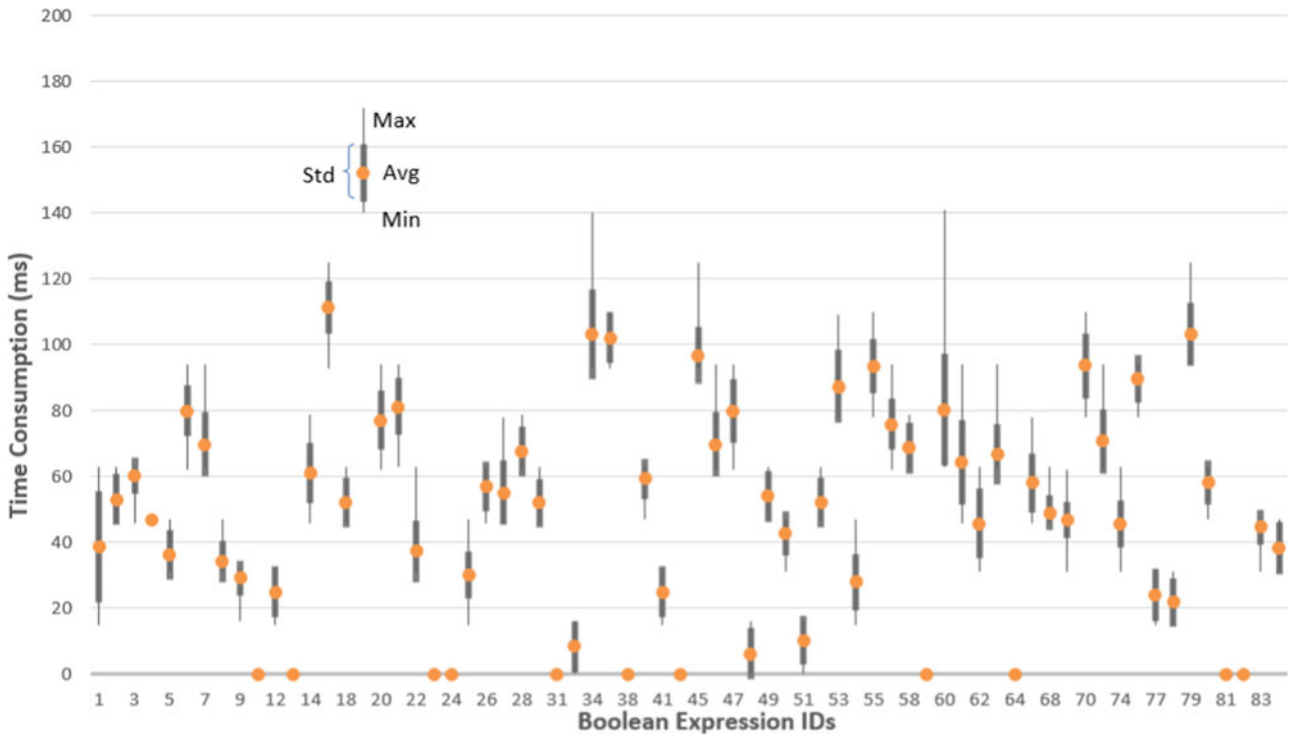


Fig. 22. Maximum, minimum, averages and standard deviations of time consumptions for some expressions.

obtains the same optimal results as cell-covering approach does.

The optimal model in [19] is based on minimal-MUMCUT that does not guarantee to detect all the double faults, in turn, it does not guarantee to detect all the double faults. Table 3 shows the empirical comparison of double-fault detection capabilities between cell-covering (CC) approach and optimal minimal-MUMCUT (OMM) [19] with the 20-case expressions, where LIF $\bowtie^2$ LIF and LRF $\bowtie^2$ LRF are double faults occurred to two terms, and LIF $\bowtie^1$ LRF and LRF $\bowtie^1$ LRF to one term. For each of the double-faults, there are two columns indicating the numbers of double-faults that cannot be detected by the cell-covering, the optimal minimal-MUMCUT, and one column to indicate the total number of double-faults for each expression.

It is observed that there is no double-fault that cannot be detected by the cell-covering approach, and on the other hand, there are double-faults that cannot be detected by the optimal minimal-MUMCUT, and as large as up to 166 for LIF $\bowtie^1$ LRF of 20-#3. Some of LIF $\bowtie^2$ LIF cannot be detected

TABLE 3
Comparisions of Double-Fault Detection Capabilities between Cell-Covering and Optimal Minimal-MUMCUT in 20-Case

IDs	LIF $\bowtie^2$ LIF			LRF $\bowtie^2$ LRF			LIF $\bowtie^1$ LRF			LRF $\bowtie^1$ LRF		
	CC	OMM	Total	CC	OMM	Total	CC	OMM	Total	CC	OMM	Total
1	0	2	52	0	0	1,822	0	0	40	0	0	40
2	0	3	231	0	0	16,890	0	0	0	0	0	0
3	0	34	43,409	0	56	1,493,549	0	166	18,152	0	64	21,444
4	0	1	77	0	0	335	0	0	96	0	0	24
5	0	10	4,513	0	0	41,869	0	16	2,856	0	0	1,548
6	0	2	92	0	0	7,774	0	0	288	0	0	576
7	0	0	672	0	0	38,232	0	20	928	0	20	1,456
8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0
10	0	6	486	0	0	53,994	0	0	1,440	0	0	3,240
11	0	3	5,113	0	48	245,997	0	32	7,464	0	18	10,920
12*	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
13	0	4	5,466	0	12	23,672	0	0	3,888	0	0	1,704
14	0	5	1,071	0	0	7,691	0	0	896	0	0	368
15	0	17	7,426	0	13	68,383	0	70	3,936	0	9	1,884
16	0	38	64,052	0	58	923,027	0	148	19,672	0	16	13,860
17	0	0	1,910	0	44	51,616	0	0	3,264	0	0	3,408
18	0	4	3,007	0	49	68,600	0	8	3,360	0	0	3,120
19	0	0	216	0	4	5,400	0	0	480	0	0	480
20	0	0	4	0	0	144	0	0	0	0	0	0

* *Expression 12 in 20-case has an un-paired parenthesis, and cannot be used in the experiment.*

because the double-faults occurred to overlapping true points, and these points will never be selected by neither of MUTP, CUTPNFP and MNFP. LRF $\bowtie^2$ LRF, LIF $\bowtie^1$LRF and LRF $\bowtie^1$ LRF incur both shrinking regions and expanding regions, MUTP, CUTPNFP and MNFP may select points from these regions, but when a region is expanded and then shrunk totally due to the double-faults, the selected points cannot detect such double-faults. The differences between the cell-covering approach and the optimal minimal-MUMCUT can be summarized as follows:

(1) Cell-covering approach is based on topological structure changes of expressions, and the optimal minimal-MUMCUT is based on MUMCUT criteria that cannot guarantee to identify double-faults as discussed above.
(2) Cell-covering approach explores the characteristics of ILP constraint coefficient matrix, and decomposes the original problem into a set of independent sub-problems, such that the overall solving time is reduced.
(3) Cell-covering approach provides a hybrid solution that amalgams an optimal approach with an approximate approach based on the problem difficulties, such that one can obtain the optimal results when the problem is relatively easy to solve, otherwise attain fairly good results within a short period of time.
(4) A tool is developed to support to generate ILP constraint coefficient matrix and objective function, diagonalizes the matrix, determines the problem difficulties, solves the problem and reports the results.

With regard to double-faults in the empirical experiments, there are additional observations:

(1) With regard to the 20-case and 25-case, there is no double-fault of LIF $\bowtie^2$ LRF, ORF+ $\bowtie^2$ LRF, TOF $\bowtie^2$ LRF, LOF $\bowtie^1$ LRF that cannot be detected by the optimal minimal-MUMCUT, i.e., the probabilities of not detections for these types of double-faults are very low.
(2) The times to generate test cases to guarantee detecting both single and double-faults are longer than that to guarantee detecting single-faults only, especially, 20-#3 and 20-#16 take more time as they have more terms to deal with, up to about 2 and 3 minutes, and the rest take no more than 20 seconds.

11 Discussion

This section discusses the threats to the validity of the proposed approach. Validity can be addressed from four points of view: external, internal, construct, and conclusion validities.

Threats to *internal validity* will be true if the proposed techniques do not hold up, or the samples provided are not representative enough. The paper provides a logical presentation of various techniques and methodologies used, and links one technique to another with detailed explanation and illustration. For example, the paper has investigated the relationship between logical faults with respect to the topological structure changes of Boolean expressions in K-map; the relationship between ILP constraints and logical fault detections; the relationship between diagonalization of ILP constraint coefficient matrix and independent sub-problems; the relationship between identity matrix, row matrix, column matrix, and ILP solutions; the relationship between cell-covering problem and set covering problem; the relationship between optimal approach and approximate approach. Furthermore, all of these techniques have been automated, and extensively tested and verified since 2009. The results have been manually evaluated to be sure that the tool [51] is working properly.

Threats related to *external validity* come from inappropriate generalization of the theory or data proposed. The proposed cell-covering approach is applicable to a variety of applications with different logical faults, that models logical faults as the changes of topological structure first, and then constructs ILP constraints associated with test cases to detect these faults, optimizes the ILP model by diagonalization of the constraint matrix, and finally solves the ILP problem with optimal techniques or approximate techniques based on the problem difficulties. Thus, the threat to external validity will be the inability of proposed approach to detect faults in real applications. But this has been demonstrated in our empirical study based on two set of Boolean expressions from two real systems: one set with 20 expressions was originally extracted from the source code of TCAS II, an air traffic collision avoidance system by Weyuker et al. [43]; and another set with 25 expressions was collected from a marine navigation system by the team-members of this paper. In addition, 88 expressions are randomly generated based on the distributions of the number of terms, the lengths of terms, the number of literals in 20 expressions and 25 expressions, the proposed approach is applied to the 88 generated expressions to further evaluate the validity.

Threats to *construct validity* will be applicable if metrics are used in the study are not appropriate or irrelevant. The metrics used in our studies are test case sizes, time to generate test cases, and fault detections, and they are widely used and accepted as proper measurements to evaluate testing strategies [44].

Threats to *conclusion validity* will be true if the case studies will produce different results if done in another time or by different people. As the proposed approach is fully spelled out with most steps automated, thus anyone using the same data with the same scenario will produce the same results.

12 Conclusion

This paper showed the topological structures of Boolean expressions faults including single and double faults of LIFs, LOFs and LRFs in K-maps, and formally proved that these topological structures are sufficient detection conditions for these faults. This is the first time that Boolean expressions faults can be characterized and visualized in a formal model K-map.

Once the topological structure is known, detecting these faults is equivalent to cover certain cells in K-map, and the detection problem is formulated as an ILP. This paper showed the subsuming relations of cell-covering between LIFs and LRFs, and between LRFs and LOFs at the refined granularities, and this significantly reduces the number of constraints and the number of variables in the coefficient

matrix of ILP, that in turn significantly reduces the time to generate the test cases.

Many properties have been observed in the constraint coefficient matrix of the ILP as the matrix can be transformed, decomposed, and optimized for efficient execution if it has certain structure.

As the ILP is NP-complete, this paper uses a hybrid approach that combines the optimal algorithms with an approximate approach to reduce the time to generate test cases. The approximate algorithm has properties that can be linked to those properties in the constraint coefficient matrix for the ILP. It is interesting to notice that those properties (described in Section 7.3) in an optimization algorithm in one problem (cell-covering) can be linked to the properties (described in Section 8.3) in an approximate algorithm for a different problem (set-covering).

Three data sets are analyzed to evaluate the proposed approach, among them, two data sets from real projects and one data set randomly generated based on the distributions of the two data sets. The experiment results showed that the proposed approach performs well, most of time optimal solutions can be obtained quickly, and for few complex problems the approximate algorithms are used to obtain near optimal solutions.

REFERENCES

[1] P. Arcaini, A. Gargantini, and E. Riccobene, "How to optimize the use of SAT and SMT solvers for test generation of boolean expressions," *Comput. J.*, vol. 58, pp. 2900–2920, Jan. 21, 2015.

[2] P. E. Black, V. Okun, and Y. Yesha, "Mutation operators for specifications," in *Proc. 15th Automated Softw. Eng. Conf.*, 2000, pp. 81–88.

[3] T. Y. Chen and M. F. Lau, "Two test data selection strategies towards testing of boolean specifications," in *Proc. 21st Int. Comput. Softw. Appl. Conf.*, 1997, pp. 608–611.

[4] T. Y. Chen, M. F. Lau and Y. T. Yu, "MUMCUT: A fault-based strategy for testing boolean specifications," in *Proc. 6th Asia-Pacific Softw. Eng. Conf.*, 1999, Art. no. 606.

[5] T. Y. Chen and M. F. Lau, "Test case selection strategies based on boolean specifications," *Softw. Test. Verif. Reliab.*, vol. 11, no. 3, pp. 165–180, Sep. 2001.

[6] J. J. Chilenski and S. P. Miller, "Applicability of modified condition/decision coverage to software testing," *Softw. Eng. J.*, vol. 9, no. 5, pp. 193–200, 1994.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT press, 2009.

[8] R. A. DeMillo, R. J. Lipton, F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *IEEE Comput.*, vol. 11, no. 4, pp. 34–41, Apr. 1978.

[9] D.-Z. Du, K.-I Ko, and X. Hu, *Design and Analysis of Approximation Algorithms [M]*. Berlin, Germany: Springer, 2011.

[10] K. A. Foster, "Sensitive test data for logic expressions," *ACM SIGSOFT Softw. Engin. Notes 9*, vol. 2, pp. 120–125, 1984.

[11] P. G. Frankl and E. J. Weyuker, "Provable improvements on branch testing," *IEEE Trans. Softw. Eng.*, vol. 19, no. 10, pp. 962–975, Oct. 1993.

[12] G. Fraser and A. Gargantini, "Generating minimal fault detecting test suites for boolean expressions," in *Proc. 3rd Int. Conf. Softw. Testing Verification Validation Workshops*, Apr. 2010, pp. 37–45.

[13] A. Gargantini, "Dealing with constraints in boolean expression testing," in *Proc. 3rd Workshop Constraints Softw. Testing Verification Anal.*, Mar. 25, 2011, pp. 322- -327.

[14] F. Gray, "Pulse code communication," U.S. Patent 2 632 058, filed Nov. 13, 1947, issued Mar. 17, 1953.

[15] R. G. Hamlet, "Testing programs with the aid of a compiler," *IEEE Trans. Softw. Eng.*, vol. 8, no. 4, pp. 371–379, 1982.

[16] R. W. Hamming, "Error detecting and error correcting codes," *Bell Syst. Techn. J.*, vol. 26, no. 2, pp. 147–160, 1950.

[17] G. Kaminski, G. Williams, and P. Ammann, "Reconciling perspectives of logic testing for software," *Softw. Testing. Verification Rel.*, vol. 18, no. 3, pp. 149–188, 2008.

[18] G. Kaminski and P. Ammann, "Using logic criterion feasibility to reduce test set size while guaranteeing fault detection," in *Proc. Int. Conf. Softw. Testing Verification Validation*, 2009, pp. 356–365.

[19] G. Kaminski and P. Ammann, "Applications of optimization to logic testing," in *Proc. Softw. Testing, Verification Validation Workshops*, 2010, pp. 331–336.

[20] G. Kaminski and P. Ammann, "Reducing logic test set size while preserving fault detection," *Softw. Testing, Verification Rel.*, vol. 21, pp. 155–193, 2011.

[21] G. Kaminski, U. Praphamontripong, P. Ammann, and J. Offutt, "A logic mutation approach to selective mutation for programs and queries," *Inf. Softw. Technol.*, vol. 53, pp. 1137–1152, 2011.

[22] M. Karnaugh, "The map method for synthesis of combinational logic circuits," *Trans. Amer. Inst. Electr. Eng. Part I*, vol. 72, no. 9, pp. 593–599, Nov. 1953.

[23] K-map, aka. Karnaugh Map. [Online]. Available: http://en.wikipedia.org/wiki /Karnaugh_map

[24] N. Kobayashi, T. Tsuchiya, and T. Kikuno, "Non-specification-based approaches to logic testing for software," *Inf. Softw. Technol.*, vol. 44, no. 2, pp. 113–121, 2002.

[25] D. R. Kuhn, "Fault classes and error detection capability of specification-based testing," *ACM Trans. Softw. Eng. Method.*, vol. 8, no. 4, pp. 411–424, 1999.

[26] M. F. Lau and Y. T. Yu, "An extended fault hierarchy for specification-based testing," *ACM Trans. Softw. Eng. Methodology*, vol. 14, no. 3, pp. 247–276, Jul. 2005.

[27] M. F. Lau, Y. Liu, and Y. T. Yu, "On the detection conditions of double faults related to terms in boolean expressions," in *Proc. 30th Annu. Int. Comput. Softw. Appl. Conf.*, 2006, pp. 403–410.

[28] M. F. Lau, Y. Liu, and Y. T. Yu, "Detecting double faults on term and literal in boolean expressions," in *Proc. 7th Int. Conf. Quality Softw.*, 2007, pp. 117–126.

[29] M. F. Lau, Y. Liu, and Y. T. Yu, "On the detection conditions of double faults related to literals in boolean expressions," in *Proc. 12th Int. Conf. Reliable Softw. Technol.–Ada-Europe*, Jun. 2007, pp. 55–68.

[30] L. J. Morell, "A theory of fault-based testing," *IEEE Trans. Softw. Eng.*, vol. 16, no. 8, pp. 844–857, Aug. 1990.

[31] K. G. Murty, *Linear Programming*. New York, NY, USA: John Wiley & Sons Inc., 1983.

[32] D. J. Richardson and M. C. Thompson, "An analysis of test data selection criteria using the RELAY model of fault detection," *IEEE Trans. Softw. Eng.*, vol. 19, no. 6, pp. 533–553, Jun. 1993.

[33] I. Stamelos, "Detecting associative shift faults in predicate testing," *J. Syst. Softw.*, vol. 66, pp. 57–63, 2003.

[34] K. C. Tai and H. K. Su, "Test generation for boolean expressions," in *Proc. 11th Int. Comput. Softw. Appl. Conf.*, 1987, pp. 278–284.

[35] K. C. Tai, M. A. Vouk, A. Paradkar, and P. Lu, "Evaluation of a predicate-based software testing strategy," *IBM Syst. J.*, vol. 33, no. 3, pp. 445–457, 1994.

[36] K. C. Tai, "Theory of fault-based predicate testing for computer programs," *IEEE Trans. Softw. Eng.*, vol. 22, no. 8, pp. 552–562, Aug. 1996.

[37] W. T. Tsai, X. Wei, Y. Chen, R. Paul, and B. Xiao, "Swiss cheese test case generation for web services testing," *IEICE Trans. Inf. Syst.*, vol. E88-D, no. 12, pp. 2691–2698, Dec. 2005.

[38] T. Tsuchiya and T. Kikuno, "On fault classes and error detection capability of specification based testing," *ACM Trans. Softw. Eng. Method.*, vol. 11, no. 1, pp. 58–62, Jan. 2002.

[39] L. Yu, W. Zhao, X. Fan, and J. Zhu, "Exploring topological structure of boolean expressions for test data selection," in *Proc. 3rd IEEE Int. Symp. Theoretical Aspects Softw. Eng.*, 2009, pp. 267–274.

[40] L. Yu, W. T. Tsai, W. Zhao, and J. Zhu, "Towards selecting test data using topological structure of boolean expressions," in *Proc. 9th Int. Conf. Quality Softw.*, 2009, pp. 31–40.

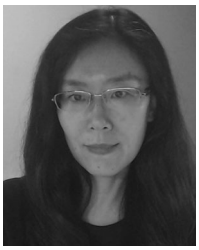
[41] Y. T. Yu and M. F. Lau, "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions," *J. Syst. Softw.*, vol. 79, no. 5, pp. 577–590, May 2006.

[42] M. A. Vouk, K. C. Tai, and A. Paradkar, "Empirical studies of predicate-based software testing," in *Proc. Int. Symp. Softw. Rel. Eng.*, 1994, pp. 55–64.

[43] E. Weyuker, T. Goradia, and A. Singh, "Automatically generating test data from a boolean specification," *IEEE Trans. Softw. Eng.*, vol. 20, no. 5, pp. 353–363, May 1994.

[44] S. N. Weiss, "What to compare when comparing test data adequacy criteria," *Software Eng. Notes*, vol. 14, pp. 42–49, Oct. 1989.

[45] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge, U.K.: Cambridge University Press, 2011.
[46] H. Zhu, "A formal analysis of the subsume relation between software test adequacy criteria," *IEEE Trans. Softw. Eng.*, vol. 22, no. 4, pp. 248–255, Apr. 1996.
[47] Set Covering. [Online]. Available: http://en.wikipedia.org/wiki/Set_cover_problem
[48] GLPK: GNU linear programming kit (GLPK). [Online]. Available: https://www.gnu.org/software/glpk
[49] Lingo 11.0, LinDo Systems Inc. [Online]. Available: http://www.lindo.com/index.php?option=com_content&view=article&id=151
[50] J. E. Mitchell, "Branch-and-cut algorithms for combinatorial optimization problems," *Handbook of Applied Optimization*. New York, NY, USA: Oxford Univ. Press, 2002, pp. 65–77.
[51] Cell-Covering Support Tool, Github. [Online]. Available: https://github.com/PKU-YU/Cell-Covering-Tool



Lian Yu received the bachelor's and master's degrees from Tsinghua University, in 1986 and 1989, respectively, and the PhD degree in information science from Yokohama National University, in 1999. She had been a researcher with Arizona State University from 2000 to 2005. Her interested research area includes software verification and validation, formal methods, and distributed computing. Currently, she is an associate professor in the School of Software and Microelectronics of Peking University. She is a member of the IEEE.

Wei-Tek Tsai received the SB degree in computer science and engineering from MIT, Cambridge, Massachusetts, in 1979, and the PhD and MS degrees in computer science from the University of California at Berkeley in 1982 and 1986, respectively. He is currently a Thousand-Talent Program professor at Beihang University, Beijing, China. He is also professor emeritus with Arizona State University, Tempe, Arizona. He has worked on a number of topics in software engineering, computer networks, service computing, and blockchains. Currently, he is the director of Digital Society and Blockchain Laboratory, Beihang University. He is a member of the IEEE.