Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 15, no. 10, October 2019, pp. 2709-2717
DOI: 10.23940/ijpe.19.10.p17.27092717

Comparing Minimal Failure-Causing Schema and Probabilistic Failure-Causing Schema on Boolean Specifications

Ziyuan Wang[a,b,*], Xueyao Li[a], Yang Li[a], and Yuqing Dai[c]

[a]*School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing, 210023, China*
[b]*Tongda College, Nanjing University of Posts and Telecommunications, Yangzhou, 225127, China*
[c]*School of Information Technology, Nanjing University of Chinese Medicine, Nanjing, 210023, China*

Abstract

Both the model of minimal failure-causing schema (MFS) and the model of probabilistic failure-causing schema (PFS) were proposed to describe characteristics of failure test cases in input-domain testing. To improve the efficiency of software debugging, input variables that are related to failure-causing schemas should be closer to the real fault-relevant input variables. In order to examine which model (MFS or PFS) can help software engineers localize fault-relevant input variables more preciously, we conduct an experiment on general-form Boolean specifications extracted from the well-known TCAS system. For each mutant of a general-form Boolean expression, the set of input variables localized by the MFSs, the set of input variables localized by the PFSs, and the set of actual input variables involved in the fault are compared. Experimental results suggest that the MFS model usually has an advantage in terms of recall, while the PFS model usually has an advantage in terms of precision. Overall, the latter has a slight advantage in terms of f-measure.

Keywords: input-domain testing; combinatorial testing; minimal failure-causing schema; probabilistic failure-causing schema; fault localization

(Submitted on August 23, 2019; Revised on September 21, 2019; Accepted on October 20, 2019)

1. Introduction

Input-domain testing techniques (sometimes referred to as black-box testing) focus on generating or selecting test cases from input domains of programs under testing, without considering codes or inner structures of programs [1]. Once failure test cases are reported during the running of test cases, software engineers need to fix the bug. Fault localization techniques can help localize the fault to improve the efficiency of debugging [2]. Differing from the code-level fault localization techniques that indicate which code structure may be faulty, input-level fault localization techniques aim to identify the characteristics of failure test cases. Both the model of minimal failure-causing schema (MFS) [3] and the model of probabilistic failure-causing schema (PFS) [4] were proposed to describe such characteristics of failure test cases.

In a failure-causing schema, partial input variables have fixed input values while other variables do not. Test cases that contain a failure-causing schema must be failure test cases (in the MFS model) or have higher probabilities to be failure test cases (in the PFS model). Failure-causing schemas obtained by input-level fault localization algorithms classify the input variables into two types: failure-causing schemas-related ones and others. To improve the efficiency of debugging, input variables that are related to failure-causing schemas should be closer to the real fault-relevant input variables. However, the effectiveness of MFS or PFS has not been studied in-depth yet. Therefore, it is doubtful whether the MFS model and the PFS model can describe characteristics of failure test cases preciously.

In order to examine which model can help localize fault-relevant input variables more preciously, we conduct an experiment on general-form Boolean specifications extracted from a well-known TCAS system in this paper. In such an experiment, for each mutant of a general-form Boolean expression, the set of input variables related to the MFSs, the set of input variables related to the PFSs, and the set of actual input variables involved in the fault are compared. Experimental

* Corresponding author.
E-mail address: wangziyuan@njupt.edu.cn

results suggest that the MFS model usually has advantages in terms of *recall*, while the PFS model usually has advantages in terms of *precision*. Overall, regarding *f-measure*, the PFS model has advantages over the MFS model in most cases.

The rest of this paper is organized as follows: preliminaries are introduced in Section 2. Experimental designs are described in Section 3. Experimental results are illustrated in Section 4. Threats to validity are analyzed in Section 5. Finally, conclusions and future works are given in Section 6.

2. Preliminaries

2.1. Boolean-Specification Testing

A Boolean expression is a string that consists of some Boolean input variables, logic operators '$\wedge$' (AND), '$\vee$' (OR), '$\neg$' (NOT), and the brackets '(' and ')'. A general-form Boolean expression means that the Boolean expression is neither a conjunction normal form nor a disjunction normal form. The goal of Boolean-specification testing is to detect faults in Boolean expressions, which are usually extracted from predicate statements of programs.

There are usually ten types of faults that should be considered in the field of Boolean-specification testing [5]:

- `ASF`: an associative shift fault is caused by omission of a pair of brackets.
- `CCF`: a clause conjunction fault is caused by replacing an occurrence of an input variable c by ($c \wedge c'$), where c' could be any possible input variable or its negation.
- `CDF`: a clause disjunction fault is caused by replacing an occurrence of an input variable c by ($c \vee c'$), where c' could be any possible input variable or its negation.
- `ENF`: an expression negation fault is caused by replacing a sub-expression by its negation.
- `LNF`: a literal negation fault is caused by replacing an occurrence of an input variable by its negation. It may sometimes be called a variable negation fault.
- `LRF`: a literal reference fault is caused by replacing an occurrence of an input variable by another variable or its negation. It may sometimes be called a variable reference fault.
- `MLF`: a missing literal fault is caused by omitting an occurrence of an input variable. It may sometimes be called a missing variable fault.
- `ORF`: an operator reference fault is caused by replacing an occurrence of a logic connective $\wedge$ (or $\vee$) by $\vee$ (or $\wedge$).
- `SA0`: a stuck-at-0 fault is caused by replacing an occurrence of an input variable by the logic constant `FALSE` (0).
- `SA1`: a stuck-at-1 fault is caused by replacing an occurrence of an input variable by the logic constant `TRUE` (1).

2.2. Minimal Failure-Causing Schema

The model of minimal failure-causing schema was proposed by Nie et. al. [3]. Consider a program under test with n input variables $F = \{f_1, f_2, \cdots, f_n\}$, in which each input variable f_i has a set of input values $V_i = \{1, 2, \cdots, a_i\}$ for $i = 1, 2, \cdots, n$ respectively. The input domain of such a program is $D = V_1 \times V_2 \times \cdots \times V_n$.

Definition 1 (Test case). A test case for such a program is an n-tuple ($v_1 \in V_1, V_2 \in V_2, \cdots, v_n \in V_n$).

Definition 2 (Schema). A k-value schema s is an n-tuple $s = (-, \cdots, -, v_{i,1}, -, \cdots, -, v_{i,2}, -, \cdots, -, v_{i,k}, -, \cdots, -)$ where $1 \le k \le n$ and $v_{i,1} \in V_{i,1}$, $v_{i,2} \in V_{i,2}$, $\cdots$, $v_{i,k} \in V_{i,k}$. There are k variables whose values have been fixed, as well as n-k variables whose values have not been fixed and are denoted as "-".

Definition 3 (Child schema and parent schema). Given a k_1-value schema $s_1 = (v_1, v_2, \cdots, v_n)$ and a k_2-value schema $s_2 = (v'_1, v'_2, \cdots, v'_n)$, if $k_1 \le k_2$ and $(v_i = -) \vee (v_i = v'_i)$ is always true for $i = 1, 2, \cdots, n$, then s_1 is a child schema of s_2 and s_2 is a parent schema of s_1. Such a relationship can be denoted as $s_1 \prec s_2$.

Definition 4 (Failure-causing schema). A k-value schema s is a failure-causing schema if, for any possible test cases $t \in D = V_1 \times V_2 \times \cdots \times V_n$, $s \prec t \Rightarrow$ (t is a failure test case).

Definition 5 (Minimal failure-causing schema). A failure-causing schema s is a minimal failure-causing schema if, for any $s' \prec s$, $s' \neq s \Rightarrow$ (s' is not a failure-causing schema).

For example, consider a Boolean expression $a \wedge (\neg b \vee \neg c) \wedge d \vee e$ and its clause disjunction faulty version $a \wedge (\neg b \vee \neg c) \wedge d \vee (d \vee e)$. There is a total of $2^5 = 32$ test cases in the input domain, where only five are failure test cases (see Table 1). By extracting child schemas from all five failure test cases and filtering them by other 27 passed test cases, 11 failure-causing schemas are obtained (see Table 2).

Table 1. All failure test cases for the faulty version $a \wedge (\neg b \vee \neg c) \wedge d \vee (d \vee e)$

	a	*b*	*c*	*d*	*e*
$test_1$	False	False	False	True	False
$test_2$	False	False	True	True	False
$test_3$	False	True	False	True	False
$test_4$	False	True	True	True	False
$test_5$	True	True	True	True	False

Table 2. All failure-causing schemas for the faulty version $a \wedge (\neg b \vee \neg c) \wedge d \vee (d \vee e)$

	a	*b*	*c*	*d*	*e*
$schema_1$	False	False	False	True	False
$schema_2$	False	False	True	True	False
$schema_3$	False	True	False	True	False
$schema_4$	False	True	True	True	False
$schema_5$	True	True	True	True	False
$schema_6$	-	True	True	True	False
$schema_7$	False	-	False	True	False
$schema_8$	False	-	True	True	False
$schema_9$	False	False	-	True	False
$schema_{10}$	False	True	-	True	False
$schema_{11}$	False	-	-	True	False

There are two minimal failure-causing schemas: (-, True, True, True, False) and (False, -, -, True, False). This suggests that all five Boolean input variables *a*, *b*, *c*, *d*, and *e* are involved in such a fault.

2.3. Probabilistic Failure-Causing Schema

The model of probabilistic failure-causing schema was proposed by Wang et. al. [4]. This model focuses on schemas that are likely to cause failures and calculates their probabilities of causing failures as well as their probabilities of appearing in testing.

Definition 6 (Failure-causing probability). A schema *s* is a probabilistic failure-causing schema with a failure-causing probability p_{fail} if the ratio of the number of failure test cases that contain *s* to the number of test cases that contain *s* is p_{fail}.

Definition 7 (Coverage probability). A schema *s* is a probabilistic failure-causing schema with a coverage probability p_{cov} if the ratio of the number of test cases that contain *s* to the number of all possible test cases in the input domain is p_{cov}.

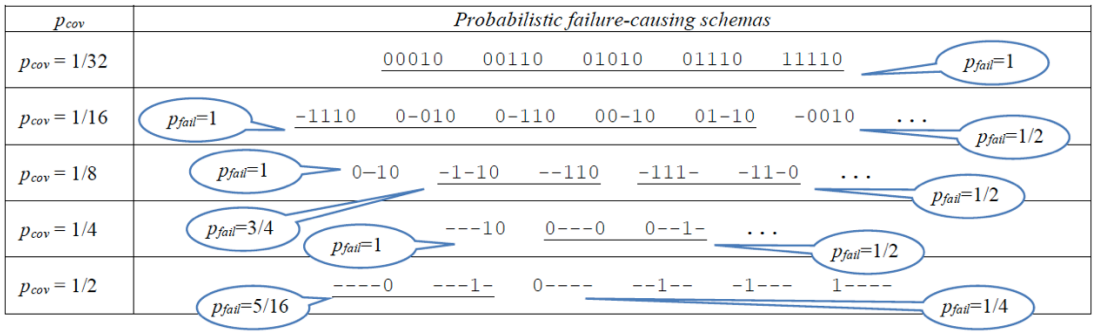
A probabilistic failure-causing schema with a failure-causing probability p_{fail} means that, for an arbitrary test case $t \in D$ that contains such a schema, the probability of the event that *t* is a failure test case is p_{fail}. The coverage probability p_{cov} means that, for an arbitrary test case $t \in D$, the probability of the event that *t* contains such a schema is p_{cov}. For the faulty Boolean expression $a \wedge (\neg b \vee \neg c) \wedge d \vee (d \vee e)$ whose original version is $a \wedge (\neg b \vee \neg c) \wedge d \vee e$, partial probabilistic failure-causing schemas as well as their failure-causing probabilities and coverage probabilities are shown in Table 3.

In the original version of the model of probabilistic failure-causing schema, the schemas with the greatest metric score, defined as $p_{fail} \times p_{cov}$, should be selected [4]. Three schemas (-, -, -, True, False), (-, -, -, True, -), and (-, -, -, -, False) are the results, which suggests that only two Boolean input variables *d* and *e* are involved in the fault.

3. Experimental Setup

This section describes the experiment to compare the model of minimal failure-causing schema and the model of probabilistic failure-causing schema on Boolean specifications.

Table 3. Partial probabilistic failure-causing schemas for the faulty version $a \wedge (\neg b \vee \neg c) \wedge d \vee (d \vee e)$



3.1. Research Questions

Both the model of minimal failure-causing schema (MFS) and the model of probabilistic failure-causing schema (PFS) aim to describe the characteristics of failure test cases preciously. Based on these models, input-domain fault localization methods output minimal or probabilistic failure-causing schemas and classify all the input variables into two classes: failure-causing schemas-related input variables and others. Input variables related to the failure-causing schemas should be closer to the real fault-relevant input variables.

To measure how effectively the MFS model or PFS model localize fault-relevant input variables, we compare the set of input variables related to the MFSs ($VarSet_{mfs\text{-}related}$) or the set of input variables related to the PFSs ($VarSet_{pfs\text{-}related}$) to the set of actual input variables that are involved in the fault ($VarSet_{relevant}$). We use the metrics *recall*, *precision*, and *f-measure* in our experiment. Consequently, the following research questions should be answered:

Research Question 1 (*Recall*): Which model can classify input variables with a higher *recall*?

Research Question 2 (*Precision*): Which model can classify input variables with a higher *precision*?

Research Question 3 (*F-measure*): Which model can classify input variables with a higher *f-measure*?

The metrics *recall*, *precision*, and *f-measure* are defined using the notations in the field of classification. In such a classification task, the set of input variables related to the MFSs or PFSs is denoted as $VarSet_{fs\text{-}related}$. True positive (*TP*) means the input variables related to MFSs or PFSs are truly fault-relevant, false negative (*FN*) means the input variables not related to MFSs or PFSs are fault-relevant, and false positive (*FP*) means the input variables related to MFSs or PFSs are not fault-relevant [6].

$$recall = \frac{TP}{TP + FN} = \frac{|VarSet_{fs\text{-}related} \cap VarSet_{relevant}|}{|VarSet_{relevant}|} \tag{1}$$

$$precision = \frac{TP}{TP + FP} = \frac{|VarSet_{fs\text{-}related} \cap VarSet_{relevant}|}{|VarSet_{fs\text{-}related}|} \tag{2}$$

$$f\text{-}measure = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{3}$$

It is evident that higher *recall*, *precision*, and *f-measure* values indicate greater effectiveness of the MFS model or the PFS model, as well as a more precious description of characteristics of failure test cases.

3.2. Experimental Subject

In the field of Boolean-specification testing, 20 general-form Boolean expressions named `TCAS1` - `TCAS20`, which were extracted from a TCAS system [7], have been widely used as benchmarks [8]. They have often been used in the experiments of input-domain testing as well [9-10]. For these 20 general-form Boolean expressions, we use ten mutation operators to create 24,521 mutants, of which 19,131 mutants are non-equivalent ones. There is only one fault in each mutant.

The set of actual input variables involved in a fault can be described as follows:

- Fault-relevant input variables for `ASF`: all the variables between the missed pair of brackets.
- Fault-relevant input variables for `CCF`: all the variables in the inserted clause conjunction.
- Fault-relevant input variables for `CDF`: all the variables in the inserted clause disjunction.
- Fault-relevant input variables for `ENF`: all the variables in the negated sub-expression.
- Fault-relevant input variables for `LNF`: the variable that can be replaced by its negation.
- Fault-relevant input variables for `LRF`: the variable that can be replaced and its substitute.
- Fault-relevant input variables for `MLF`: the missed variable.
- Fault-relevant input variables for `ORF`: the variables that can be connected by the replaced logic operator.
- Fault-relevant input variables for `SA0`: the variable that can be replaced by the logic constant `FALSE` (0).
- Fault-relevant input variables for `SA1`: the variable that can be replaced by the logic constant `TRUE` (1).

3.3. Experimental Process

There is a total of three steps in the experiment:

(1) For each fault, find all the actual input variables involved in such a fault according to the rules given in the previous sub-section to form the set of real fault-relevant input variables $VarSet_{relevant}$.

(2) For each fault, find all the minimal failure-causing schemas to form the set of input variables related to all the MFSs $VarSet_{mfs\text{-}related}$, and calculate the *recall*, *precision*, and *f-measure* scores of $VarSet_{mfs\text{-}related}$ and $VarSet_{relevant}$. Here, the failure-causing schemas are obtained according to the process described in reference [11] and using the tool described in reference [12].

(3) For each fault, find all the probabilistic failure-causing schemas with the greatest score $p_{fail} \times p_{cov}$ to form the set of input variables related to the greatest-score PFSs $VarSet_{pfs\text{-}related}$, and calculate the *recall*, *precision*, and *f-measure* scores of $VarSet_{mfs\text{-}related}$ and $VarSet_{relevant}$. The approach is similar to that utilized in the second step.

4. Experimental Results

This section illustrates the results of comparing the MFS model and the PFS model in terms of *recall*, *precision*, and *f-measure*. Findings and remarks are discussed based on these results.

4.1. Results for RQ1

Figures 1 and 2 compare the *recall* score of the set of input variables related to the MFSs and the *recall* score of the set of input variables related to the greatest-score PFSs for each fault. There are 20 groups of box graphs in Figure 1, where each group stands for the mutants obtained from one of the 20 original Boolean expressions. In each group of box graphs, the left box graph illustrates the *recall* scores for the PFS model, while the right box graph illustrates the *recall* scores for the MFS model.

Similarly, there are ten groups of box graphs in Figure 2, where each group stands for the mutants obtained by performing one of ten mutation operators. *Recall* scores are compared in each group.

4.2. Results for RQ2

Figures 3 and 4 compare the *precision* score of the set of input variables related to the MFSs and the *precision* score of the set of input variables related to the greatest-score PFSs for each fault. There are 20 groups of box graphs in Figure 3, where each group stands for the mutants obtained from one of the 20 original Boolean expressions. In each group of box graphs, the left one illustrates the *precision* scores for the PFS model, while the right one illustrates the *precision* scores for the MFS model.

Similarly, there are ten groups of box graphs in Figure 4, where each group stands for the mutants obtained by performing one of ten mutation operators. *Precision* scores are compared in each group.

4.3. Results for RQ3

Figures 5 and 6 compare the *f-measure* score of the set of input variables related to the MFSs and the *f-measure* score of the set of input variables related to the greatest-score PFSs for each fault. There are 20 groups of box graphs in Figure 5, where

each group stands for the mutants obtained from one of the 20 original Boolean expressions. In each group of box graphs, the left one illustrates the *f-measure* scores for the PFS model, while the right one illustrates the *f-measure* scores for the MFS model.

Similarly, there are ten groups of box graphs in Figure 6, where each group stands for the mutants obtained by performing one of 10 mutation operators. *F-measure* scores are compared in each group.

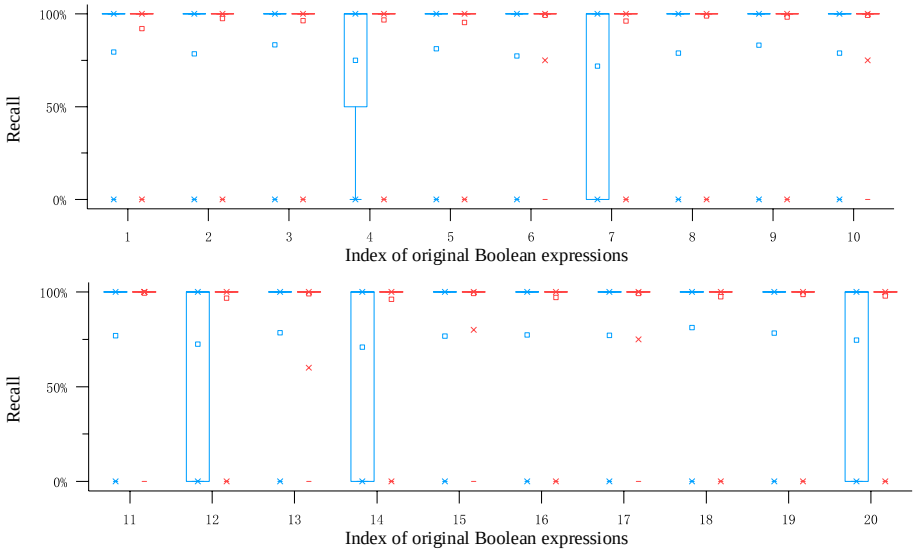


Figure 1. *Recall* scores of the sets of input variables related to the MFSs and the high-score PFSs (for faults in 20 original Boolean expressions)

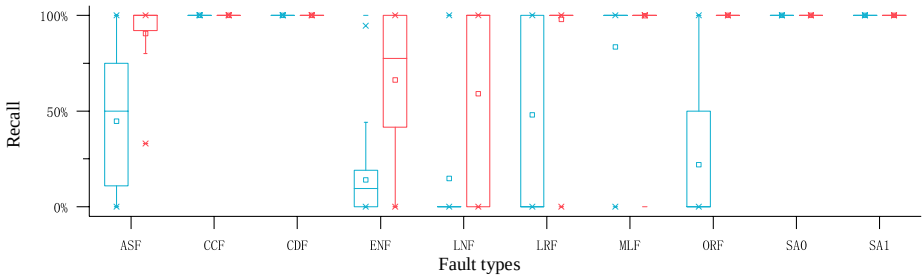


Figure 2. *Recall* scores of the sets of input variables related to the MFSs and the high-score PFSs (for faults with ten fault types)

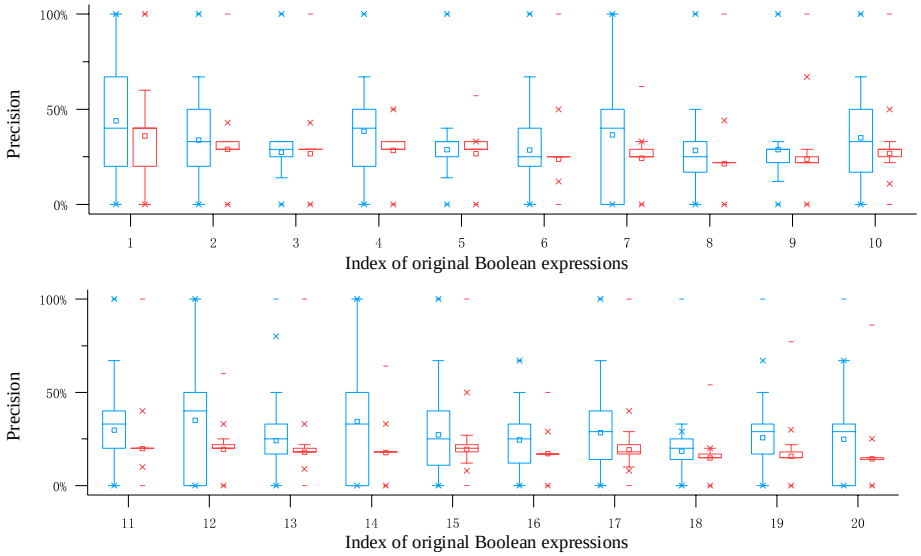


Figure 3. *Precision* scores of the sets of input variables related to the MFSs and the high-score PFSs (for faults in 20 original Boolean expressions)

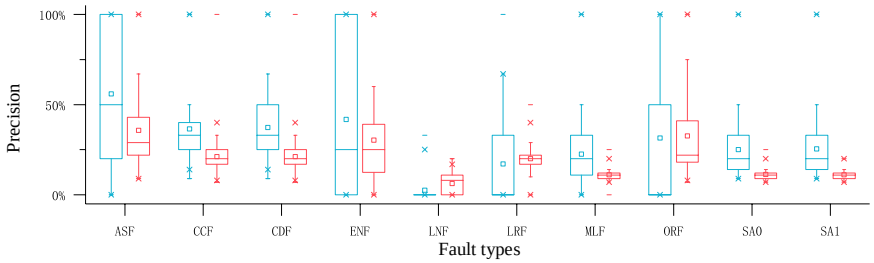


Figure 4. *Precision* scores of the sets of input variables related to the MFSs and the high-score PFSs (for faults with ten fault types)

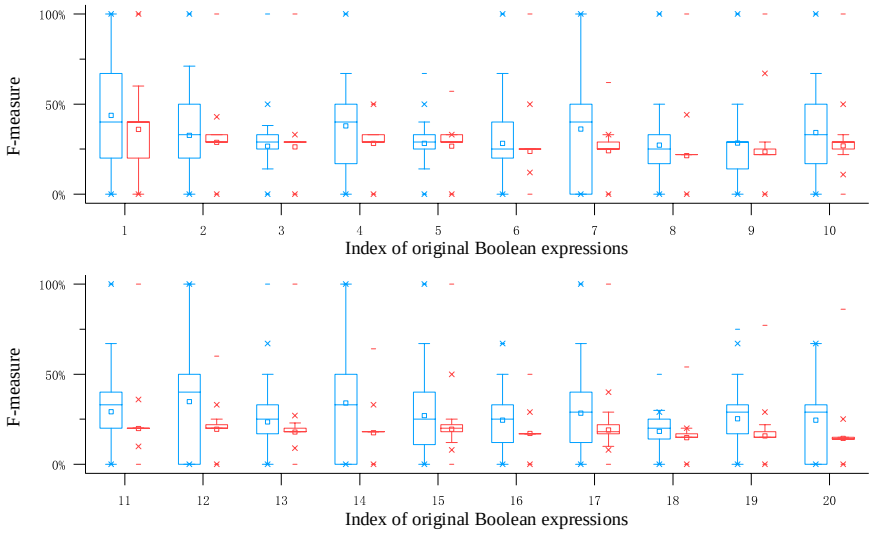


Figure 5. *F-measure* scores of the sets of input variables related to the MFSs and the high-score PFSs (for faults in 20 original Boolean expressions)

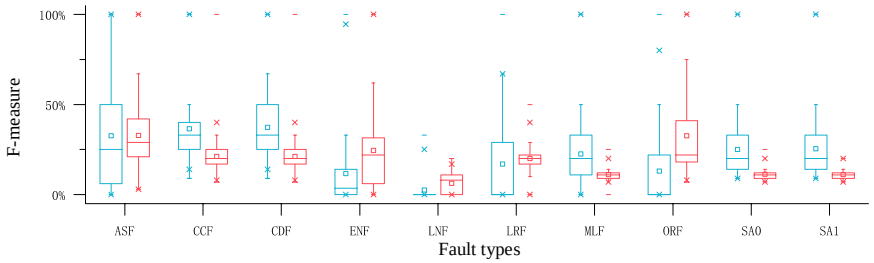


Figure 6. *F-measure* scores of the sets of input variables related to the MFSs and the high-score PFSs (for faults with ten fault types)

4.4. Finding and Remarks

The experimental results show the following: (1) the MFS model usually has significant advantages in aspect of *recall* for the `ASF`, `ENF`, `LNF`, `LRF`, and `ORF` faults in `TCAS4`, `TCAS7`, `TCAS12`, `TCAS14`, and `TCAS20`. (2) The PFS model usually has significant advantages in terms of *precision* for the `ASF`, `CCF`, `CDF`, `MLF`, `SA0`, and `SA1` faults in almost all the Boolean expressions except `TCAS2`, `TCAS3`, `TCAS5`, and `TCAS6`. (3) In terms of *f-measure*, the MFS model has advantages for the `ENF`, `LNF`, `LRF`, and `ORF` faults, while the PFS model has advantages for the `CCF`, `CDF`, `MLF`, `SA0`, and `SA1` faults. Overall, the PFS model has advantages over the MFS model for faults in almost all the Boolean expressions except `TCAS2`, `TCAS3`, `TCAS5`, and `TCAS6`.

These results suggest that the PFS model is more competitive than the MFS model in the scenario of input-level fault localization. The reasons include both the advantages in terms of *precision* and the advantages in terms of *f-measure*.

5. Threats to Validity

Threats to internal validity are concerned with the uncontrolled factors that may influence the experimental results. The first uncontrolled factor is the correctness of the set of fault-relevant input variables or the set of failure-causing schema-related

input variables. In our experiment, actual fault-relevant input variables for each fault are set according to the definition of the corresponding fault type. Minimal and probabilistic failure-causing schemas for each fault are obtained by an open-sourced tool [11-12]. All the intermediate results regarding fault-relevant input variables, MFS-related input variables, and PFS-related input variables are tested to avoid any possible incorrectness. Another uncontrolled factor involves the different sets of test cases that are utilized to find MFSs and PFSs and may lead to different results. In our experiment, for each Boolean expression, the exhaustive test set is executed and analysed to identify the real MFSs and PFSs for each fault.

Threats to external validity are concerned with whether the results in experiments are generalizable. In our experiment, experimental subjects include 20 general-form Boolean expressions extracted from the TCAS system and 19,131 non-equivalent mutants generated by ten popular mutation operators in the field of Boolean-specification testing. These experimental subjects have been widely used as benchmarks in the field of Boolean-specification testing. It is reasonable that they could represent realistic cases for comparing the combinatorial testing technique and the random testing technique.

6. Conclusions and Future Works

Both the MFS model and the PFS model were proposed to describe characteristics of failure test cases. In order to examine which model (MFS or PFS) can help software engineers debug programs more efficiently, we designed an experiment on Boolean specifications to compare which model can localize fault-relevant input variables more preciously. Experimental results indicated the MFS model usually has significant advantages in terms of *recall*, while the PFS model usually has significant advantages in terms of both *precision* and *f-measure*. This means that the PFS model is more competitive than the MFS model in the scenario of input-level fault localization.

Although significant results regarding the comparison of the MFS model and the PFS model have been presented in this paper, there are still many works required in the future. (1) Some other experimental subjects except Boolean specifications should be taken into consideration in the comparison between the MFS model and the PFS model to generalize the results concluded in this paper. (2) Many input-level fault localization algorithms, including non-adaptive methods [13-15] and adaptive methods [16-18], have been proposed for the MFS model. Similarly, PFS-based algorithms should be also developed to localize high-score PFSs effectively and efficiently. (3) In the initial version of the PFS model, the metric score is defined as the product of failure-causing probability and coverage probability. More approaches to measure and select PFSs are required in the future.

Acknowledgements

This work is supported by the National Nature Science Foundation of China (No. 61772259) and the Foundation of Nanjing University of Posts and Telecommunications (No. NY219079).

References

1. P. Bourque and R. E. Fairley, "SWEBOK: Guide to the Software Engineering Body of Knowledge (v3.0)," IEEE Computer Society, 2014
2. A. Zeller and R. Hildebrandt, "Simplifying and Isolating Failure-Inducing Input," *IEEE Transaction on Software Engineering*, Vol. 28, No. 2, pp. 183-200, 2002
3. C. Nie and H. Leung, "The Minimal Failure-Causing Schema of Combinatorial Testing," *ACM Transactions on Software Engineering and Methodology*, Vol. 20, No. 4, pp. 15, 2011
4. Z. Wang, Y. Qi, and J. Lin, "Probabilistic Failure-Causing Schema in Input-Domain Testing," in *Proceedings of 27th International Conference on Software Engineering and Knowledge Engineering* (*SEKE2015*), pp. 726, 2015
5. Z. Chen, T. Y. Chen, and B. Xu, "A Revisit of Fault Class Hierarchies in General Boolean Specifications," *ACM Transactions on Software Engineering and Methodology*, Vol. 20, No. 3, pp. 13, 2011
6. D. L. Olson and D. Delen, "Advanced Data Mining Techniques," Springer Berlin Heidelberg, 2008
7. N. G. Leveson, M. P. E. Heimdahl, H. Hildreth, and J. D. Reese, "Requirements Specification for Process-Control Systems," *IEEE Transaction Software Engineering*, Vol. 20, No. 9, pp. 684-707, 1994
8. E. Weyuker, T. Goradia, and A. Singh, "Automatically Generating Test Data from a Boolean Specification," *IEEE Transaction Software Engineering*, Vol. 20, No. 5, pp. 353-363, 1994
9. N. Kobayashi, T. Tsuchiya, and T. Kikuno, "Application of Non-Specification-based Approaches to Logic Testing for Software," in *Proceedings of International Conference on Dependable Systems and Networks* (*DSN2001*), pp. 337-346, 2001
10. N. Kobayashi, T. Tsuchiya, and T. Kikuno, "Non-Specification-based Approaches to Logic Testing for Software," *Information and Software Technology*, Vol. 44, No. 2, pp. 113-121, 2002
11. Z. Wang and Y. Qi, "Why Combinatorial Testing Works: Analyzing Minimal Failure-Causing Schemas in Logic Expressions," in *Proceedings of IEEE 8th International Conference on Software Testing, Verification, and Validation Workshops* (ICSTW2015): *4th International Workshop on Combinatorial Testing* (IWCT2015), April 13-17, 2015

12. Z. Wang and M. Yu, "BoolMuTest: A Prototype Tool for Fault-based Boolean-Specification Testing," in *Proceedings of 30th International Conference on Software Engineering and Knowledge Engineering* (*SEKE2018*), pp. 720-721, 2018
13. C. J. Colbourn and D. W. McClary, "Locating and Detecting Arrays for Interaction Faults," *Journal of Combinatorial Optimization*, Vol. 15, No. 1, pp. 17-48, 2008
14. C. Martinez, L. Moura, and D. Panario, "Locating Errors Using ELAs, Covering Arrays, and Adaptive Testing Algorithms," *Siam Journal on Discrete Mathematics*, Vol. 23, No. 4, pp. 1776-1799, 2010
15. T. Konishi, H. Kojima, and H. Nakagawa, "Finding Minimum Locating Arrays using a SAT Solver," in *Proceedings of IEEE International Conference on Software Testing, Verification and Validation Workshops* (*ICSTW2017*), 2017
16. Z. Wang, B. Xu, and L. Chen, "Adaptive Interaction Fault Location based on Combinatorial Testing," in *Proceedings of International Conference on Quality Software* (QSIC2010), pp. 495-502, 2010
17. Z. Zhang and J. Zhang, "Characterizing Failure-Causing Parameter Interactions by Adaptive Testing," in *Proceedings of the 2011 International Symposium on Software Testing and Analysis* (*ISSTA2011*), pp. 331-341, 2011
18. X. Niu, C. Nie, and Y. Lei, "Identifying Failure-Inducing Combinations using Tuple Relationship," in *Proceedings of IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops* (*ICSTW2013*), pp. 271-280, 2013