Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 15, no. 11, November 2019, pp. 2952-2961
DOI: 10.23940/ijpe.19.11.p14.29522961

Fault Detection Capabilities of Combinatorial Testing and Random Testing for Boolean-Specifications

Ziyuan Wang[a,b,c,*], Yang Li[a], Xueqing Gu[a], Xiaojia Zheng[a], and Min Yu[a]

[a]*School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing, 210023, China*
[b]*Software Testing Engineering Laboratory of Jiangsu Province, Nanjing, 211169, China*
[c]*Tongda College, Nanjing University of Posts and Telecommunications, Yangzhou, 225127, China*

Abstract

The problem of fault detection capability of combinatorial testing has drawn a lot of attention. People conducted many experiments on different subjects to compare fault detection capabilities of combinatorial testing and random testing. However, previous confusing results can hardly answer the question of whether combinatorial test suite detects more faults than random test suite. To answer this question more trustfully, we conducted an experiment on general-form Boolean-specifications. In our experiment, fault detection frequencies and fault detection ratios in combinatorial testing, where test suites are generated by some classic combinatorial test generation algorithms, are collected by the repeated running of combinatorial testing. Moreover, fault detection probabilities in random testing are obtained by theoretical analysis. By comparing fault detection frequencies, ratios, and probabilities, our experimental results suggest that combinatorial testing has a little advantage of fault detection capability over random testing.

Keywords: combinatorial testing; random testing; fault detection probability; fault detection frequency; fault detection ratio

(Submitted on August 10, 2019; Revised on October 10, 2019; Accepted on November 15, 2019)

1. Introduction

One of the most important goals of software testing is to detect as many faults as possible. All the testing managers want to find a software testing technique with higher fault detection capacity in their software projects because they need a testing technique that detects more faults than others. As one of the input domain testing techniques, combinatorial testing has been widely utilized in applications, since it was reported that many failures were triggered by the combinations of parametric values. Some people believe that combinatorial testing has a high fault detection capacity, while some others doubt that. To answer the question about the fault detection capacity of combinatorial testing, we need to compare combinatorial testing with some baseline software testing techniques, e.g., random testing.

Both random testing and combinatorial testing belong to input domain testing techniques [1]. Many people designed experiments to answer the question of whether the combinatorial test suite has a higher fault detection capacity than the random test suite containing the same number of test cases. Some studies reported positive results that combinatorial testing was more effective than random testing [2-5], while some other studies reported negative results [6-7]. Most of the studies that gave positive results focused on Boolean-specification testing [2-4]. Although these results on Boolean-specification testing provided partial confidences in combinatorial testing, there is still a limitation that only partial fault types in the field of Boolean-specification testing were taken into consideration.

To compare the fault detection capacities of combinatorial testing and random testing, we perform mutation testing on general-form Boolean-specifications in this paper. The innovations in our experiment include: (1) We used more mutation operators to generate mutants; (2) We used several classic combinatorial test generation algorithms to generate combinatorial test suites; (3) We used more metrics in our experiment: fault detection frequencies and fault detection ratios are counted in the repeated running of combinatorial testing, while fault detection probabilities in random testing are

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

obtained by theoretical analysis. By comparing fault detection frequencies, ratios, and probabilities, our experimental results suggest combinatorial testing has a little advantage of fault detection capability over random testing.

The rest of this paper is organized as follows. Some preliminaries are introduced in Section 2. Research questions and experimental designs are described in Section 3. Experimental results are illustrated in Section 4. Threats to validity are analyzed in Section 5. Related works are presented in Section 6. Finally, conclusions are given in Section 7.

2. Preliminaries

Some preliminaries, including Boolean-specification testing, combinatorial testing, and random testing, will be presented.

2.1. Boolean-Specification Testing

A Boolean expression is a string that includes some Boolean input variables, logic operators '$\wedge$' (AND), '$\vee$' (OR), '$\neg$' (NOT), and the brackets '(' and ')'. Boolean-specification testing aims to detect faults in Boolean expressions extracted from predicate statements of programs. Many previous works pay attention to the conjunction normal form (CNF) or disjunction normal form (DNF) Boolean expressions. However, software engineers never write CNF or DNF Boolean expressions in programs. Therefore, we mainly focus on general-form Boolean expressions in this paper.

There are ten fault types usually considered in fault-based Boolean-specification testing [8]:

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

`ASF`, `CCF`, `CDF`, `ORF`, `ENF` are core fault types [8]. This means that a test suite that detects all the faults with these five core fault types could detect all the faults with the other five fault types too.

2.2. Combinatorial Testing

Suppose there is a testing subject with n input variables $F = \{f_1, f_2, \cdots, f_n\}$, where 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. If the cardinalities of the sets of input values for all the input variables are uniform ($a_1 = a_2 = \cdots = a_n = a$), the testing subject is a fixed-level system with an input model $M = \{a^n\}$ or an a-level system. Otherwise, the testing subject is a mixed-level system with an input model $M = \{a_1 \times a_2 \times \cdots \times a_n\}$.

Definition 1 A τ-way fixed-level covering array $CA(m; \tau, n, a)$ is an $m \times n$ array on totally a symbols with the property that each $m \times \tau$ sub-array contains all possible τ-tuples from these a symbols at least once. Here, τ is called the strength of such a fixed-level covering array [9].

For a given testing subject with input model $M = \{a^n\}$, we could obtain a τ-way combinatorial test suite from a τ-way a-level covering array by mapping each row in the covering array to a test case in the combinatorial test suite. In this paper, we will perform the combinatorial testing technique on Boolean-specifications, where each input variable has two input values `FALSE`(0) and `TRUE`(1), so we will pay attention to 2-level covering arrays. E.g., for a general-form Boolean expression with 5 Boolean input variables, a 2-way combinatorial test suite could be found in Table 1.

Table 1. 2-way combinatorial test suite for input model $M = \{2^5\}$

No.	Input 1	Input 2	Input 3	Input 4	Input 5
1	False	False	False	True	False
2	False	False	True	False	True
3	False	True	False	False	False
4	True	False	True	False	True
5	True	True	False	True	True
6	True	True	True	True	False

The combinatorial test suite with fewer test cases consumes a lower cost of the software testing process. Therefore, people proposed many combinatorial test generation algorithms to generate a combinatorial test suite that is as small as possible. Most of combinatorial test generation algorithms could be classified into algebraic methods [10-11], greedy heuristic methods based on one-test-at-a-time [12] and in-parameter-order strategy [13], and meta-heuristic search methods [14], etc. Especially for 2-level systems, Maity et al. and Nie et al. proposed two similar algorithms to generate optimal 2-way 2-level combinatorial test suites [15-16]. Kleitman et al. proved that the size of a 3-way 2-level combinatorial test suite is equal to or greater than $3.212 \times \log(n \times (1 + o(1)))$ [17]. Naor et al. proved that the size of an optimal τ-way 2-level combinatorial test suite is less than or equal to $2^{\tau} \times \tau^{O(\log(\tau))} \times \log(n)$ [18].

2.3. Random Testing

In scenarios where the input domain of the testing subject is known, random testing technique picks points randomly within the input domain space to form the random test cases for the testing subject [1]. All the test cases within the input domain have the same probabilities to be selected in random testing. E.g., for a general-form Boolean expression with 5 Boolean input variables, its input domain should contain 2^5 test cases. The probability of the event that an arbitrary test case is picked up in random testing is $1/2^5$.

We can calculate the fault detection probability of a random test suite with a given number of test cases if we have the failure rate of a given fault in the testing subject. Let n be the number of test cases in a random test suite (n is smaller than the size of input domain), and θ be the failure rate of a given fault ($0 < \theta < 1$). The fault detection probability of a random test suite for such a fault is the probability of the event that at least one randomly selected test case detects such a fault:

$$F\text{-}Probability = 1 - (1 - \theta)^n \tag{1}$$

Where the failure rate of a given fault in the testing subject is the ratio of the number of test cases that detect such a fault to the number of test cases in the input domain:

$$\theta = \frac{\text{Number of test cases that detect such a fault}}{\text{Number of test cases in input domain}} \tag{2}$$

3. Experimental Setup

This section describes our experiment to compare fault detection capabilities of combinatorial testing and random testing on general-form Boolean-specifications.

3.1. Research Questions

To compare fault detraction capabilities of the combinatorial test suite and the random test suite with the same size, we will use two metrics called *fault detection frequency* and *fault detection ratio*, respectively. Consequently, we need to answer the following research questions in our experiment.

Research Question 1 (fault detection frequency): In a large number of runnings of combinatorial testing and random testing, for each fault, is there a significant difference between the frequency of such a fault detected by combinatorial test suites and the frequency of such a fault detected by random test suites?

Research Question 2 (fault detection ratio): In each running of combinatorial testing and random testing, for a group of faults, is there a significant difference between the ratio of the faults detected by the combinatorial test suite and the ratio of the faults detected by the random test suite?

The testing technique with higher fault detection frequency and higher fault detection ratio is the one that has better fault detection capability.

3.2. Experimental Subject

We take 20 general-form Boolean expressions extracted from the TCAS system [19] as experimental subjects. For each Boolean expression, we generate mutants by using all the ten mutation operators. There are 24521 mutants, and 19131 of them are non-equivalent mutants. These original Boolean expressions and their mutants have been widely utilized as benchmarks in the field of Boolean-specification testing [20]. By running all possible test cases in the input domain of each original Boolean-specification, we can identify all the failed test cases for each mutant. The failure rates of all the 19131 faults in 19131 non-equivalent mutants are shown in Figure 1.

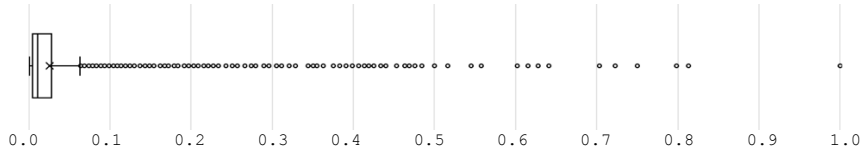

Figure 1. Distribution of failure rates of 19131 mutants generated by ten mutation operators from 20 original Boolean expressions

For each non-equivalent mutant, the failure-triggering fault interaction (FTFI) number, which was defined by Kuhn et al. [21], are counted and illustrated in Figures 2 and 3. The higher the FTFI number is, the more difficult a fault is to be detected in combinatorial testing. The data of FTFI numbers are obtained in our previous works [22].

Figure 2. FTFI numbers of mutants generated from 20 original Boolean Expressions

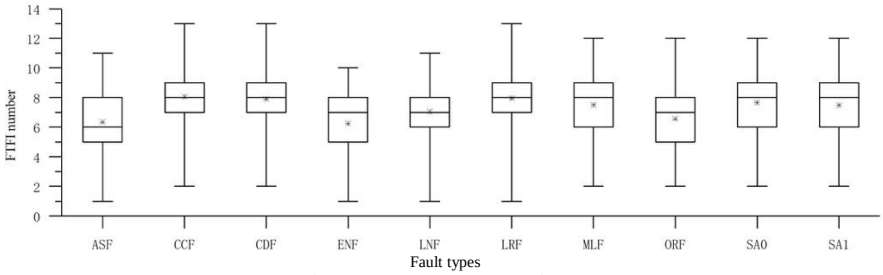

Figure 3. FTFI numbers of mutants generated by performing ten mutation operators

3.3. Experimental Process

There are three steps in our experiment: (1) generating combinatorial test suites, (2) collecting fault detection frequencies and ratios in combinatorial testing, and (3) calculating fault detection probabilities of random test suites.

Step 1 Combinatorial test suites are generated by some classic combinatorial test generation algorithms including Greedy [23], DDA [12], IPO [13], DensityRO [24], and ReqMerge [25]. For each original Boolean expression, an algorithm will generate 100 different τ-way combinatorial test suites to form a group of combinatorial test suites for τ = 2, 3, 4, respectively. Note that some algorithms (including Greedy, DDA, IPO, and DensityRO) are deterministic algorithms that always output the same results for the same inputs. Therefore, when running an algorithm, some randomly generated seeding test cases will be assigned to output the rich diversity combinatorial test suites in a large number of runnings.

Step 2 To answer the first research question, in a large number of the running of combinatorial testing, for each mutant,

the frequency of such a fault detected by different groups of combinatorial test suites should be collected. In order to answer the second research question, in each running of combinatorial testing, for a group of mutants obtained from the same original Boolean expression or obtained by performing the same mutation operator, the ratio of the number of mutants killed by each combinatorial test suite to the total number of mutants in such a group should be collected.

Step 3 In random testing, we calculate theoretical fault detection probability at this step. According to the principle of frequentism, "the probability could be defined as the limit of its relative frequency in a large number of trials"[1]. Therefore, we can compare the theoretical fault detection probability with the actual fault detection frequency. Besides, by the law of large numbers, "the sample averages converge in probability and almost surely to the expected value with the increase of the number of trails"[2]. Therefore, we can compare the mean value of fault detection probabilities with the mean value of actual fault detection ratios. According to Formula 1, the fault detection probability of random testing depends on the number of test cases. When comparing random testing and τ-way combinatorial testing, for each Boolean expression, we set the size of the random test suite as the minimum values of the sizes of τ-way combinatorial test suites generated by different algorithms. Table 2 shows the sizes of combinatorial test suites generated by algorithms Greedy, DDA, IPO, DensityRO, and ReqMerge, respectively. We can find from Table 2 that algorithm Greedy and DensityRO generate almost all the smallest combinatorial test suites.

Table 2. Sizes of combinatorial test suites generated by different combinatorial test generation algorithms (including Greedy, DDA, IPO, DensityRO, and ReqMerge) for each Boolean expression

Number of input variables	Original expression	Sizes of 2-way combinatorial test suites	Sizes of 3-way combinatorial test suites	Sizes of 4-way combinatorial test suites
5	TCAS1	6/6/6/6/10	12/12/12/12/16	24/16/16/16/28
7	TCAS2, TCAS3, TCAS4, TCAS5	7/7/8/7/14	16/16/16/15/24	29/36/36/32/48
8	TCAS6, TCAS7	8/8/10/8/16	17/16/18/15/28	33/34/38/32/56
9	TCAS8, TCAS9, TCAS10	8/8/10/8/18	17/18/20/18/32	35/37/42/41/71
10	TCAS11, TCAS12	9/9/12/9/20	16/18/20/18/36	36/40/44/40/84
11	TCAS13, TCAS14	9/9/12/9/22	19/20/20/19/40	49/45/46/43/89
12	TCAS15, TCAS16, TCAS17	9/9/14/9/24	18/21/22/19/44	43/46/52/47/100
13	TCAS18, TCAS19	10/9/14/9/26	20/20/22/20/48	46/49/52/50/110
14	TCAS20	10/9/16/9/28	22/22/24/21/52	47/52/52/51/117

4. Experimental Results

Experimental results are illustrated in Figures 4 to 11. In Figures 4 to 7, we compare fault detection probabilities in random testing and actual fault detection frequencies in combinatorial testing. In Figures 8 to Figure 11, we compare the mean value of fault detection probabilities in random testing and actual fault detection ratios in combinatorial testing.

4.1. Results for RQ1

In Figures 4 to 6, there are 20 groups of box-graphs in each figure in which each group stands for the mutants obtained from one of 20 original Boolean expressions. Similarly, in each sub-figure of Figure 7, there are ten groups of box-graphs, where each group stands for the mutants obtained by performing one of ten mutation operators. In each group of box-graphs, the first box-graph illustrates theoretical fault detection probabilities of random test suite for all the mutants obtained from the same original Boolean expression, and the next five illustrate actual frequencies of each mutant detected by a group of τ-way combinatorial test suites generated by Greedy, DDA, IPO, DensityRO, and ReqMerge, respectively.

4.2. Results for RQ2

In Figures 9 to 11, there are 20 groups of box-graphs in each figure, in which each group stands for the mutants obtained from one of 20 original Boolean expressions. Similarly, in Figure 8, there are ten groups of box-graphs in each sub-figure, where each group stands for the mutants obtained by performing one of ten mutation operators. In each group of box-graphs, the first box-graph illustrates the mean value of theoretical fault detection probabilities of random test suite for all the mutants obtained from the same original Boolean expression, and the next five illustrate actual ratios of these mutants detected by 100 different τ-way combinatorial test suites generated by Greedy, DDA, IPO, DensityRO, and ReqMerge, respectively.

[1] https://en.wikipedia.org/wiki/Frequentist_probability
[2] https://en.wikipedia.org/wiki/Law_of_large_numbers

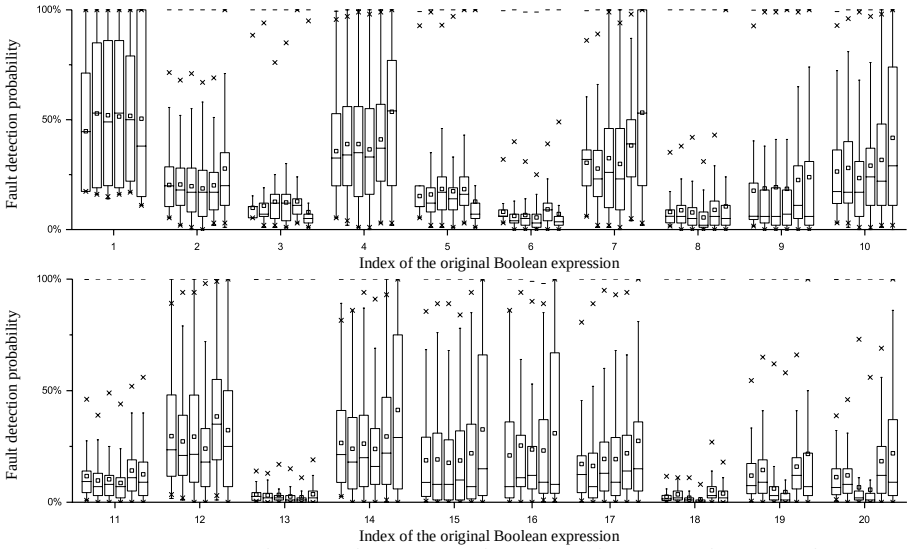

Figure 4. Fault detection probabilities of RTs and fault detection frequencies of 2-way CTs for mutants from 20 Boolean Expressions

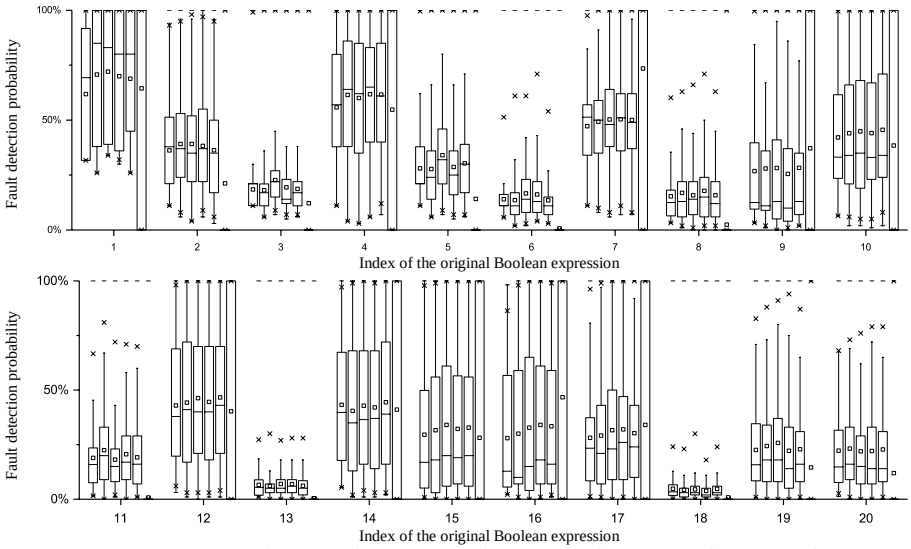

Figure 5. Fault detection probabilities of RTs and fault detection frequencies of 3-way CTs for mutants from 20 Boolean Expressions

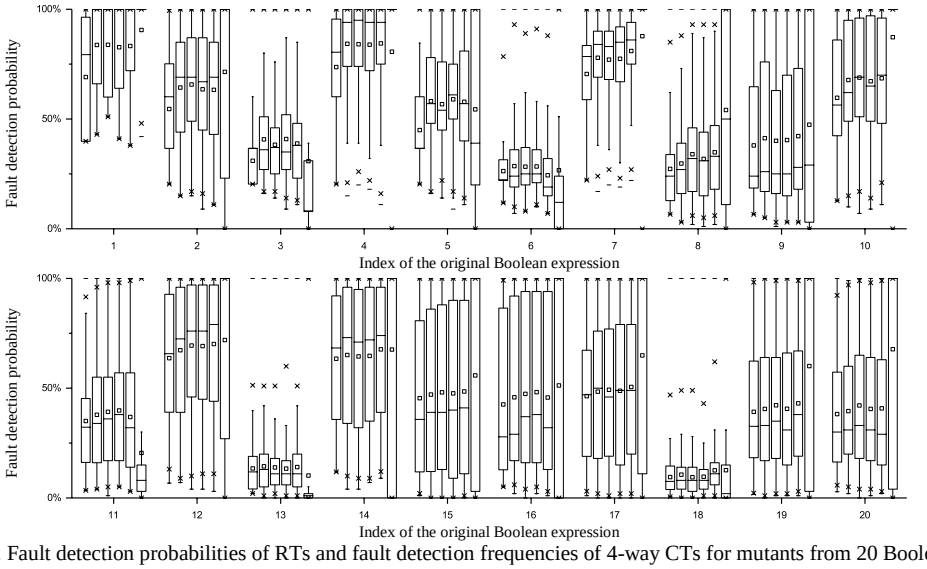

Figure 6. Fault detection probabilities of RTs and fault detection frequencies of 4-way CTs for mutants from 20 Boolean Expressions

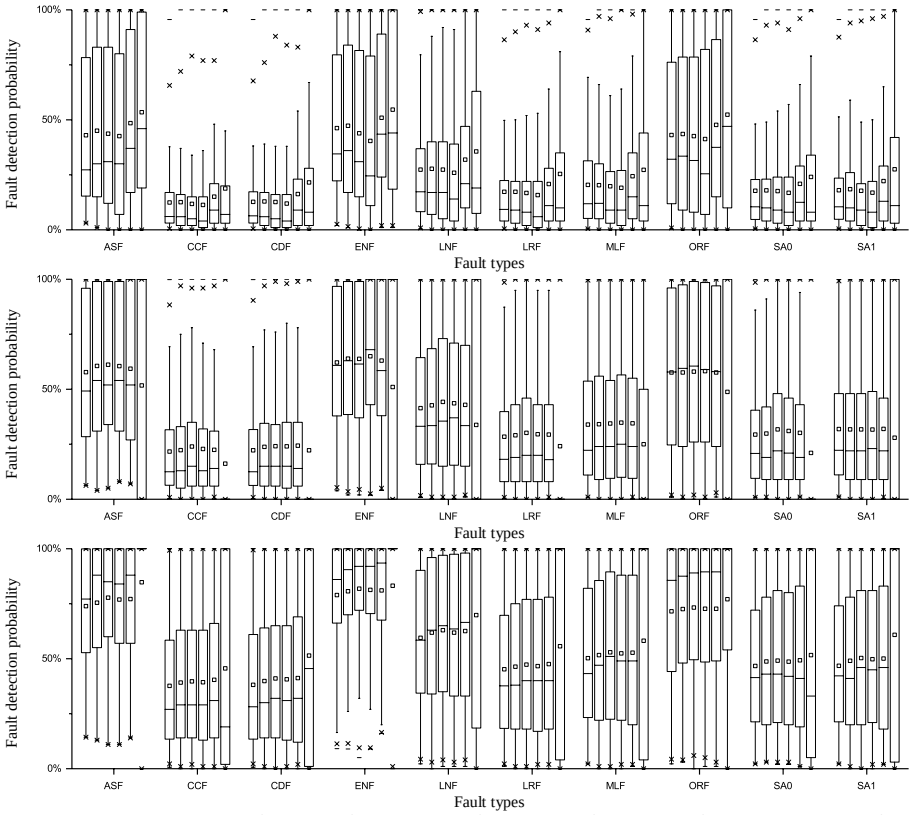

Figure 7. Fault detection probabilities of RTs and fault detection frequencies of τ-way CTs for mutants with ten fault types (τ = 2, 3, 4)

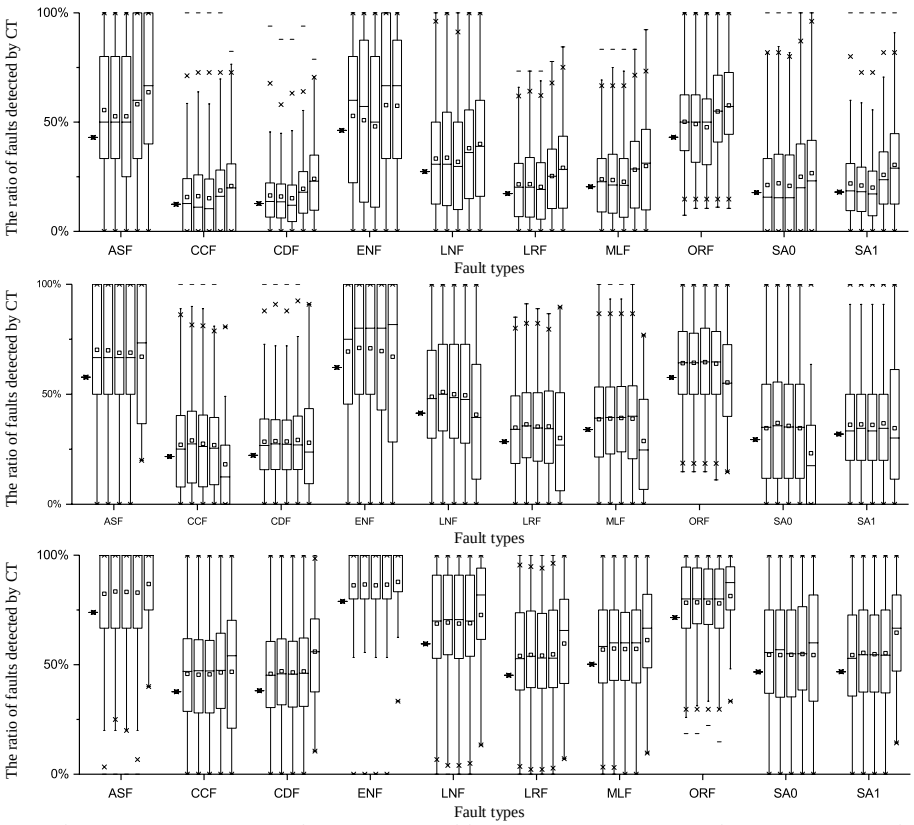

Figure 8. Average fault detection probability of RTs and ratios of faults detected by τ-way CTs for mutants with ten fault types (τ = 2, 3, 4)

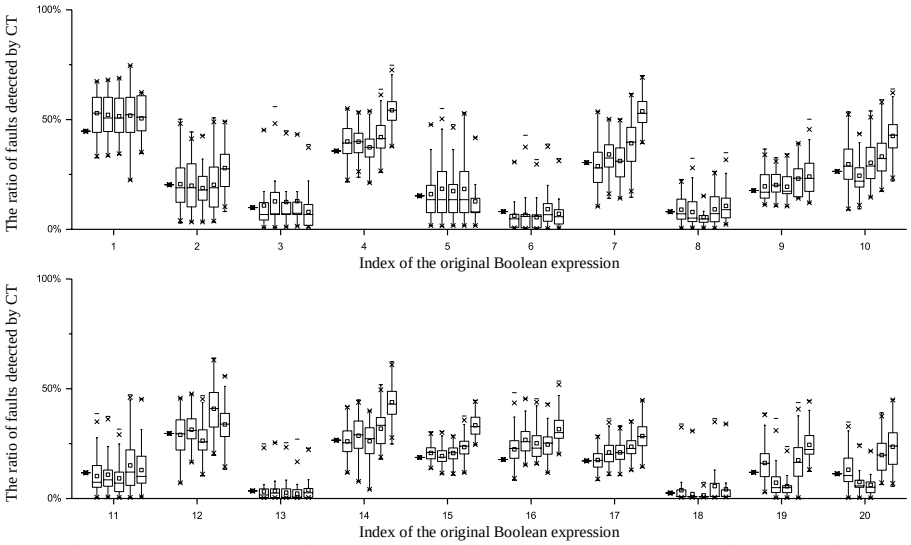

Figure 9. Average fault detection probability of RTs and ratios of faults detected by 2-way CTs for mutants from 20 Boolean expressions

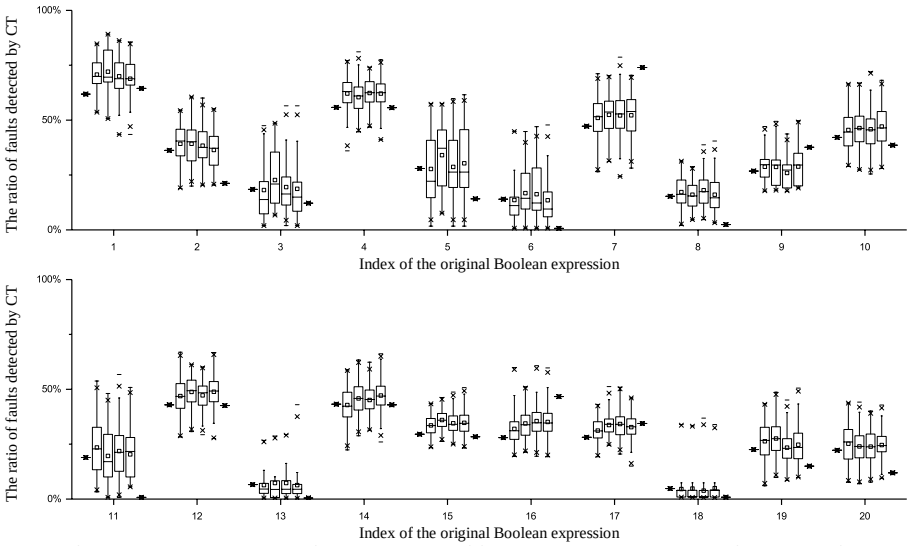

Figure 10. Average fault detection probability of RTs and ratios of faults detected by 3-way CTs for mutants from 20 Boolean expressions

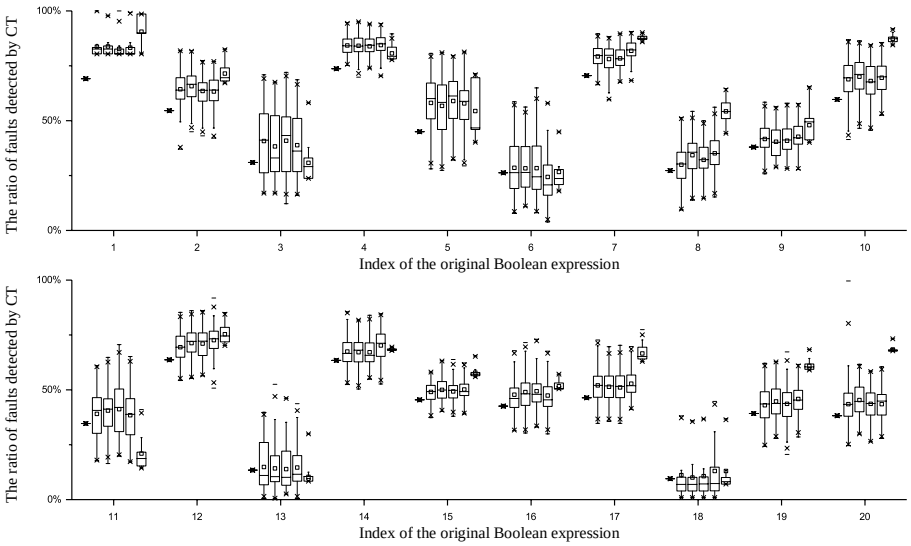

Figure 11. Average fault detection probability of RTs and ratios of faults detected by 4-way CTs for mutants from 20 Boolean expressions

4.3. Finding and Remarks

We can answer two research questions by comparing theoretical fault detection probabilities in random testing with actual fault detection frequencies and ratios in combinatorial testing. (1) There is a significant difference between the frequency of each fault detected by combinatorial test suites and the frequency of each fault detected by random test suites. (2) There is a significant difference between the ratio of a group of faults detected by each combinatorial test suite and the ratio of a group of faults detected by each random test suite.

Detailed experimental results suggest that combinatorial testing has a little advantage of fault detection capability over random testing, especially when the strength of combinatorial testing is higher. We can also find some phenomena: (1) When the dimension of the input domain (or the number of input variables) tends to be large, there are fewer differences between fault detection capacities of combinatorial testing and random testing; (2) When the FTFI number of fault tends to be large, there are fewer differences between fault detection capacities of combinatorial testing and random testing.

5. Threats to Validity

Threats to internal validity are concerned with the uncontrolled factors that may also be responsible for the results. In our experiment, fault detection frequencies and fault detection ratios were collected by running combinatorial test suites on faulty Boolean expressions. Different results may be obtained by running different combinatorial test suites with the same strength. To void the problem, we selected some well-known classic combinatorial test generation algorithms. Furthermore, we seed different seeding test cases to each algorithm to generate rich diversity combinatorial test suites.

Threats to external validity are concerned with whether the results in our experiments are generalizable for other situations. In our experiment, 20 general-form Boolean expressions extracted from the TCAS system and 19131 non-equivalent mutants generated by ten well-known mutation operators were selected as experimental subjects. These original Boolean expressions and their mutants have been widely utilized as benchmarks in the field of Boolean-specification testing. We believe that they could represent realistic cases for comparing the combinatorial testing technique and the random testing technique.

6. Related Works

Many people designed experiments to answer the question of whether the combinatorial test suite has a higher fault detection capacity than a random test suite with the same size.

Kobayashi et al. conducted an experiment on 20 Boolean expressions extracted from the TCAS system [2]. Balance et al. conducted an experiment on five groups of Boolean expressions, where each contained 20 expressions with the same numbers of input variables to the TCAS expressions [3]. Vilkomir et al. conducted an experiment on 1000 Boolean expressions [4]. All these experiments suggested that combinatorial testing has a little advantage of fault detection capability over random testing for faults with types ASF, ENF, ORF, VNF, and VRF. They omitted the other five fault types.

Schroeder et al. conducted an experiment on two real systems DMAS and LAS and indicated no significant difference between fault detection capacities of combinatorial testing and random testing [6]. Ghandehari et al. conducted an experiment on seven programs in the Siemens suite and reported that the advantage of combinatorial testing over random testing is not significant enough as expected [7]. Nie et al. conducted an experiment on nine programs, including FLEX, GREP, GZIP, SED, MAKE, NANOXML, DRUPAL, BUSYBOX, and LINUX. Their results suggested that combinatorial testing is better than random testing in many cases, especially when the failure rate is lower [5].

7. Conclusions

To compare the fault detection capacities of combinatorial testing and random testing, we conducted an experiment on 20 general-form Boolean expressions extracted from the TCAS system, which were usually used as benchmarks in the field of Boolean-specification testing. By analyzing 19131 faults with ten fault types, experimental results indicated that combinatorial testing has a little advantage of fault detection capability over random testing, especially when the strength of combinatorial testing is higher. Furthermore, the differences between the fault detection capacities of combinatorial testing and random testing become less if the dimension of the input domain or the FTFI number tends to be large.

Previous works indicated that there might be many factors, including failure rate, the dimension of the input domain, and the FTFI number, that may affect fault detection capacities of combinatorial testing and random testing. In future works, some other factors should be studied.

Acknowledgements

This work is supported by the National Nature Science Foundation of China (61772259) and the Foundation of Nanjing University of Posts and Telecommunications (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. 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
3. W. A. Ballance, S. Vikomir, and W. Jenkins, "Effectiveness of Pair-Wise Testing for Software with Boolean Inputs," in *Proceedings of IEEE 5th International Conference on Software Testing, Verification and Validation* (*ICST2012*): *1st International Workshop on Combinatorial Testing* (*IWCT2012*), pp. 580-586, 2016
4. S. Vikomir, O. Starov, and R. Bhambroo, "Evaluation of T-Wise Approach for Testing Logical Expressions in Software," in *Proceedings of IEEE 6th International Conference on Software Testing, Verification and Validation Workshops* (*ICSTW2013*), *2nd International Workshop on Combinatorial Testing* (*IWCT2013*), pp. 249-256, 2013
5. H. Wu, C. Nie, J. Petke, Y. Jia, and M. Harman, "An Empirical Comparison of Combinatorial Testing, Random Testing and Adaptive Random Testing," *IEEE Transactions on Software Engineering*, DOI: 10.1109/TSE.2018.2852744
6. P. J. Schroeder, P. Bolaki, and V. Gopu, "Comparing the Fault Detection Effectiveness of N-way and Random Test Suites," in *Proceedings of the 2004 International Symposium on Empirical Software Engineering* (*ISESE2004*), pp. 49-59, Redondo Beach, California, USA, August 19-20, 2004
7. L. S. Ghandehari, J. Czerwonka, Y. Lei, S. Shafiee, R. Kacker, and R. Kuhn, "An Empirical Comparison of Combinatorial and Random Testing," in *Proceedings of IEEE 7th International Conference on Software Testing, Verification, and Validation Workshops* (*ICSTW2014*), *3rd International Workshop on Combinatorial Testing* (*IWCT2014*), pp. 68-77, 2014
8. 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* (*TOSEM*), Vol. 20, No. 3, pp. 13, 2011
9. C. Nie and H. Leung, "A Survey of Combinatorial Testing," *ACM Computing Surveys* (*CSUR*), Vol. 43, No. 2, pp. 11, 2011
10. R. Mandl, "Orthogonal Latin Squares: An Application of Experimental Design to Compiler Testing," *Communications of the ACM*, Vol. 28, No. 10, pp. 1054-1058, 1985
11. C. J. Colbourn, S. S. Martirosyan, T. V. Trung, and R. A. Walker II, "Roux-Type Constructions for Covering Arrays of Strengths Three and Four," *Designs, Codes and Cryptography*, Vol. 41, No. 1, pp. 33-57, 2006
12. R. C. Bryce and C. J. Colbourn, "A Density-based Greedy Algorithm for Higher Strength Covering Arrays," *Software Testing, Verification and Reliability*, Vol. 19, No. 1, pp. 37-53, 2009
13. Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG IPOG-D: Efficient Test Generation for Multi-Way Combinatorial Testing," *Software Testing, Verification and Reliability*, Vol. 18, No. 3, pp. 125-148, 2008
14. J. Lin, C. Luo, S. Cai, K. Su, D. Hao, and L. Zhang, "TCA: An Efficient Two-Mode Meta-Heuristic Algorithm for Combinatorial Test Generation," in *Proceedings of 30th IEEE/ACM International Conference on Automated Software Engineering* (*ASE2015*), pp. 494-505, Lincoln, Nebraska, USA, November 9-13, 2015
15. S. Maity and A. Nayak, "Improved Test Generation Algorithms for Pair-Wise Testing," in *Proceedings of 16th IEEE International Symposium on Software Reliability Engineering* (*ISSRE2005*), November 8-11, 2005
16. C. H. Nie, B. W. Xu, and L. Shi, "A New Pairwise Covering Test Data Generation Algorithm for the System with Many 2-Level Factors," *Chinese Journal of Computers*, Vol. 29, No. 6, pp. 841-848, 2006
17. D. J. Kleitman and J. Spencer, "Families of K-Independent Sets," *Discrete Mathematics*, Vol. 6, pp. 255-262, 1973
18. M. Naor, L. J. Schulman, and A. Srinvasan, "Splitters and Near-Optimal Randomization," in *Proceedings of 36th Annual Symposium on Foundations of Computer Science* (*FOCS1996*), pp. 182-191, 1996
19. 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
20. 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
21. D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software Fault Interaction and Implication for Software Testing," *IEEE Transaction on Software Engineering*, Vol. 30, No. 6, pp. 418-421, 2004
22. 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
23. P. J. Schroeder, "Black-Box Test Reduction using Input-Output Analysis: [Dissertation for PhD]," Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA, 2001
24. Z. Wang, B. Xu, and C. Nie, "Greedy Heuristic Algorithms to Generate Variable Strength Combinatorial Test Suite," in *Proceedings of the 8th International Conference on Quality Software* (*QSIC2008*), pp. 155-160, Oxford, UK, August 12-13, 2008
25. Z. Wang, C. Nie, and B. Xu, "Generating Combinatorial Test Suite for Interaction Relationship," in *Proceedings of 4th International Workshop on Software Quality Assurance* (*SOQUA2007*), pp. 55-61, Croatia, September 3-4, 2007