Improving MC/DC and Fault Detection Strength Using Combinatorial Testing

Dong Li[1], Linghuan Hu[2], Ruizhi Gao[2], W. Eric Wong[2,*], D. Richard Kuhn[3], Raghu N. Kacker[3]
[1]China Electronic Product Reliability and Environmental Testing Research Institute, Guangzhou, China
[2]Department of Computer Science, University of Texas at Dallas, USA
[3]National Institute of Standards and Technology, Gaithersburg, MD 20899, USA
lidong@ceprei.com
{lxh131930, gxr116020, ewong}@utdallas.edu
{kuhn, raghu.kacker}@nist.gov

***Abstract*— Software, in many different fields and tasks, has played a critical role and even replaced humans to improve efficiency and safety. However, catastrophic consequences can be caused by implementation bugs and design defects. Modified condition/decision coverage (MC/DC), required by the Federal Aviation Administration on Level A (the most safety critical system), has been shown to be effective in detecting software bugs. However, generating tests to achieve high MC/DC can be very expensive and time consuming. Recently, many studies showed that combinatorial testing (CT) could generate high-quality test cases in a cost-effective way. Can CT generate test cases to achieve high MC/DC? In this paper, we conduct an empirical study on two real-life programs to evaluate the efficiency and effectiveness of using combinatorial testing to improve MC/DC coverage achievement, as well as the fault detection strength.**

Keywords—Combinatorial Testing, MC/DC Coverage, Fault Detection

I. INTRODUCTION

In the past few years, testing for safety-critical systems, such as traffic control and aircraft navigation systems, became one of the most challenging tasks in the software quality assurance field. Catastrophic accidents that result in tremendous economic damage or even human casualties can be caused by the implementation bugs and design defects of safety-critical systems. On December 20, 1995, American Airlines Flight 965 from Miami, Florida, to Cali, Colombia, crashed into a 9,800 foot mountain. This was due to a navigation software error, and resulted in 159 deaths [1].

To prevent such tragedies from occurring again, the Federal Aviation Administration requires that Level A systems are tested using test cases that can achieve modified condition/decision coverage (MC/DC) defined in DO-178B/C [2][3]. MC/DC criterion requires each condition of decision in the program to be tested by showing its independent effect on the whole decision. Although many studies have showed that MC/DC is effective and efficient in detecting software bugs [4][5], the generation of test cases to achieve high MC/DC can be very difficult and expensive. To address this issue, different techniques have been proposed for generating test cases to achieve high MC/DC. For example, one of the proposed solutions is symbolic execution-based test generation (white-box) [6][7][8], which generates test cases by conducting comprehensive symbolic execution and constraint solving. The advantages of symbolic execution-based test generation is that it can be fully automated, and in the ideal scenario, it requires practitioners almost no domain knowledge to perform the test generation. However, there are also several well-known drawbacks of current symbolic execution-based test generation techniques. First, due to the path explosion problem, symbolic execution requires large amounts of resources (computation and memory) to perform the execution analysis and constraint solving [9]. Therefore, it might not be able to generate test cases for large and complex programs, especially the ones with too many loop control flows. Second, the symbolic execution-based test generation techniques are platform and language dependent; each technique can work only for certain languages and platforms. Moreover, most of the tools for symbolic execution-based test generation are not ready for commercial application because they are not available to the public, and the quality of the tools cannot be guaranteed.

In addition to the costly white-box test generation techniques, the black-box combinatorial testing (CT) has been shown to be effective and efficient in generating test cases with high fault detection strength, and it is very easy to apply industry settings [10][11][12][13]. CT treats a system under test (SUT) as a black-box, and it focuses on the interactions among various input parameters. It uses well-developed CT algorithms [14][15] to generate a test set that covers a large amount of combinations among input parameters while keeping the test set to a small size. Therefore, it is worth conducting an empirical study to evaluate the effectiveness and efficiency of using CT to generate tests to achieve high MC/DC, as well as high fault detection strength. In this paper, we want to investigate the following three research questions:

- Can CT generate test cases to achieve high MC/DC?
- How to use CT to improve coverage achievements (statement, branch, and MC/DC) in an efficient way?
- Can tests generated using CT achieve high fault detection strength?

The remainder of paper is organized as follows: in Section II, we introduce the basic concepts of CT. We will then present our experiment setup in Section III. In section VI, experiment results and detailed discussions are included, followed by our conclusions and future work in Section V.

* Corresponding author: W. Eric Wong (Email: ewong@utdallas.edu)

978-1-5386-2072-4/17 $31.00 © 2017 IEEE
DOI 10.1109/QRS-C.2017.131

II. COMBINATORIAL TESTING

In both industry and academia, regardless of gender, culture, and experience, a consensus among almost all developers is that there are always some sophisticated bugs in a software system. Even when the software system has been tested or deployed for a long time, bugs be triggered by one or some special combinations of input values [11][12][16]. These combinations of input values are usually either too rare in the real world or counterintuitive for software developers and testers to generate the responding test cases.

A simple solution to cover these special input combinations is to perform exhaustive testing. Unfortunately, it is almost impossible to apply exhaustive testing to complex programs with a large input domain. Consider the following sample program:

Table 1. Sample Program

Functionalities	Number of Possible Values
A	T, F
B	T, F
C	T, F
D	T, F

This program has four functionalities which can be enabled during the execution. We use "T" to denote that a functionality is enabled, and "F" to indicate the opposite. To perform the exhaustive testing for this program, a test set with size of 16 (=2×2×2×2) is needed. For illustration purposes, the sample program presented in Table 1 is simple, and therefore, the number of generated exhaustive tests is small. However, for a real-world complex system, the number of exhaustive combinations can easily be larger than one thousand or even one million, which makes it almost impossible for practitioners to finish the test.

Since testing all combinations among all parameters is not practical, a practitioner can generate a test set that includes all combinations between certain parameters to reduce the size of the test set. To do that, we can use CT to generate a test set with a small size to cover all combinations between any t parameters ($t \leq$ total no. of parameters), where t is the interaction strength. We also use t-way test set to denote that it contains all combinations among t parameters [10][17]. For example, Table 3 shows the twenty-six 2-way combinations of the sample program. Intuitively, we need a relatively large test set to cover all of the twenty-six combinations. Using the IPOG algorithm, we only need six test cases, as shown in Table 2.

Table 2. A 2-way Test Set Generated Using IPOG

	A	B	C	D
t_1	T	T	F	F
t_2	T	F	T	T
t_3	F	T	T	F
t_4	F	F	F	T
t_5	F	T	F	T
t_6	T	F	F	F

Notice that different CT algorithms might generate different test sets for the same interaction strength. To achieve good performance, with respect to improving coverage achievements and fault detection strength when using CT in real-world scenarios, choosing an appropriate interaction strength is very important. We will give a detailed discussion in Section V.

Table 3. All Combinations Between Two Parameters

A = T, B = T	A = T, B = F	A = F, B = T	A = F, B = F
A = T, C = T	A = T, C = F	A = F, C = T	A = F, C = F
A = T, D = T	A = T, D = F	A = F, D = T	A = F, D = F
B = T, C = T	B = T, C = F	B = F, C = T	B = F, C = F
B = T, D = T	B = T, D = F	B = F, D = T	B = F, D = F
C = T, D = T	C = T, D = F	C = F, D = T	C = F, D = F

III. EXPERIMENT SETUP

To answer the three research questions in Section I, we conduct our empirical study on two real-world C programs with complex control flows. The first subject program is TCAS, a widely-used module of a traffic collision avoidance system [17][18]. The second subject program, PEL [19], is a constraint verification module of a customization program for a photo editing software. To generate the CT test set, we use ACTS (v2.9) [20], a GUI-based CT tool developed by National Institute of Standards and Technology (NIST). ACTS is very easy to use; the tool has the ability to generate tests with interaction strength from 2-way to 6-way, with a user-friendly GUI and a command line version suitable for use in scripts or system calls from another tool. It only takes few minutes for a practitioner to learn how to set up and generate the test sets. A screenshot of ACTS is shown in Figure 1.

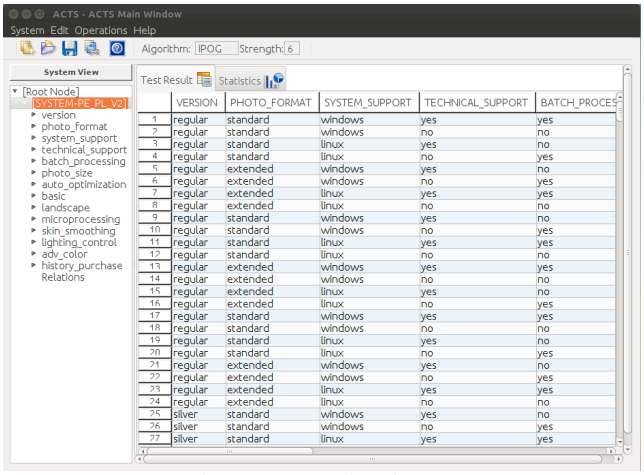


Figure 1. A screenshot of ACTS

In the experiment, we use the IPOG algorithm implemented by ACTS to generate the test sets with interaction strength from 2-way to 5-way.

A. Input Modeling and Test Generation for TCAS

We download the original implementation, and 41 mutants of TCAS from Software-artifact Infrastructure Repository [21]. The detailed description of TCAS is shown as follows:

- Size: 174 LOC
- Number of decisions: 15
- Number of input parameters: 12
- Number of output parameter: 1

In the 41 mutants (each with one seeded fault), v5 and v25 have the same faults as v27 and v39 respectively. Therefore, we arbitrarily exclude v27, and v39 from our experiment.

We reuse the input model from Kuhn's study [17]. The details of both input parameters and their corresponding values are shown in Table 4.

Table 4. Input Model for TCAS

Input Parameters	Values
Cur_Vertical_Sep	299, 300, 601
High_Confidence	0, 1
Two_of_Three_reports_Valid	0, 1
Own_Tracked_Alt	1, 2
Other_Tracked_Alt	1, 2
Own_Tracked_Alt_Rate	600, 601
Alt_Layer_Value	0, 1, 2, 3
Up_Separation	0, 399, 400, 499, 500, 639, 640, 739, 740, 840
Down_Separation	0, 399, 400, 499, 500, 639, 640, 739, 740, 840
Other_RAC	0, 1, 2
Other_Capability	1, 2
Climb_Inhibit	0, 1

Using this input model, the total number of all possible combinations of input parameters is 230,400. We generate five test sets with interaction strength from 2-way to 6-way using ACTS and IPOG algorithm, as shown in Table 5.

Table 5. Generated Test Sets for TCAS

2-way	3-way	4-way	5-way	6-way
100	400	1,359	4,233	11,021

In comparing the number of exhaustive combinations, it is very clear that the sizes of these test sets are significantly reduced, yet they still retain a certain interaction strength. In the five generated test sets, we exclude the 6-way test set as the 11,021 test cases may be too many for a real-world case.

B. Input Modeling and Test Generation for PEL

PEL is a software product line for a photo editing software that performs various photo editing tasks, such as photo batch processing, skin smoothing, advanced color adjustment, etc. We conduct our study on the constraint verification module of PEL, and the following show the detailed descriptions:

- Size: 957 LOC
- Number of decisions: 132
- Number of input parameters: 16
- Number of output parameter: 1

There are 10 distinct faulty versions, with each containing one of the real faults retrieved from the development phase. After carefully analyzing the requirements of PEL, we build our input model, shown in Table 6.

Table 6. Input Model for PEL

Input Parameters	Values
Version	Regular, silver, gold, platinum, diamond
Photo Format	Standard, extended
OS Support	Windows, MacOS, linux
Technical Support	yes, no
Batch Processing	yes, no
Max Photo Size	100MB, 300MB, 600MB, 1GB
Auto Optimization	yes, no
Basic Filter	yes, no
Landscape Filter	yes, no
Facial Filter	yes, no
Micro-processing Filter	yes, no
Skin Smoothing Plugin	yes, no
Lighting Control Plugin	yes, no
Advanced Color Plugin	yes, no
Social Network Plugin	yes, no
Accumulative Purchase Amount	100, 999, 1000, 1001, 2000, 4999, 5000, 5001, 8000, 9999, 10000, 10001, 15000

Using exhaustive testing to cover all the combinations among all input parameters, a user would need to have a test set with a size of 3,194,880. We use ACTS and IPOG algorithm to generate the following CT test sets, shown in Table 7. Notice that we also exclude the 6-way test set from our experiment.

Table 7. Generated Test Sets for PEL

2-way	3-way	4-way	5-way	6-way
65	264	851	2,659	7,667

Regenerating test sets using IPOG does not produce different results as IPOG is deterministic. An interesting observation is that, although the size of input domain of PEL is larger than that of TCAS, the number of test cases in each generated test set for PEL is smaller.

C. Coverage Collection and Fault Detection

We compile both subject programs using GCC 4.84 under Ubuntu 14.04. We use GCOV to measure both statement and branch coverage. For the MC/DC, we implement and use our own instrumentation and coverage measurement tool, MCDC-Star, for both programs. The MCDC-Star measures the MC/DC using the definition of DO-178C [3][22] (masking MC/DC) under short-circuit evaluation.

For the fault detection, we compare the outputs between correct versions and faulty versions. As every faulty version (for both subject programs) contains only one fault, the bug is considered to be detected if a faulty version produces a different output.

IV. Experiment Results

In this section, we examine the experiment results from two aspects: 1) the coverage achievements (statement, branch, and MC/DC) and 2) the fault detection strength.

A. Coverage Achievement

The coverage achievements of the test sets (2-way to 5-way) for TCAS are presented in Figure 2. In general, for each test set, the statement coverage at each data point is higher than the other two corresponding criteria, and the MC/DC is the lowest among the three coverage criteria. The three coverage achievements of each test set, from 2-way to 5-way have a significant increase (from 0% to 71.6%) within the first 15 test cases. Overall, the achievements of all three coverage criteria are improved simultaneously but with different scales, and the cumulative coverage achievements of 2-way, 3-way, and 4-way test sets increase with their corresponding interaction strength. Although the 5-way test set has 2,874 more test cases than the 4-way test set, the cumulative coverage achievement of the 5-way is the same as the 4-way test set.

In addition, to investigate the coverage achievement of each test set, we make three observation points: A, B, and C. Each point represents the moment of finishing the execution of the first 100, 400, and 1359 test cases. We choose these four points based on the sizes of generated 2-way, 3-way, and 4-way test sets. The coverage achievements of each test set at points A, B, and C are shown in Table 8.

Table 8. Coverage Achievement at Points A, B, and C for TCAS

	2-way	3-way	4-way	5-way	
A	79.01%	71.60%	71.60%	75.31%	Statement
	54.55%	45.45%	45.45%	54.55%	Branch
	51.56%	39.06%	39.06%	45.31%	MC/DC
B	N/A	87.65%	88.89%	85.19%	Statement
		83.33%	86.36%	80.30%	Branch
		75.00%	79.69%	67.19%	MC/DC
C	N/A	N/A	91.36%	87.65%	Statement
			89.39%	83.33%	Branch
			87.50%	75.00%	MC/DC

At point A, the 2-way test set achieves the highest in all three coverage criteria when compared to 3-way, 4-way, and 5-way. Note its branch coverage is the same as 5-way. The 4-way test set achieves the best results at points B and C. In general, the 5-way test set performs the worst at points A, B, and C as compared to the other three test sets.

For the PEL, the coverage achievements are shown in Figure 3. Similar to the results of TCAS, the statement coverage achieved by each test case is higher than the other two coverage criteria in most of the data points, and MC/DC achievement is still the lowest among the three coverage criteria. The coverage achievements of the three coverage criteria increase simultaneously most of the time. However, the difference between branch and MC/DC coverage achievements is not significant when compared with the result in TCAS. We carefully investigate the cause by examining both the coverage reports and the program implementations. The reason for this is that PEL has a large amount of decisions with a single condition. For such a decision, its MC/DC is very easily achievable, and the MC/DC measurement is very similar to the measurement of branch coverage in GCOV.

The coverage achievements of the three criteria of all the test sets are improved significantly (from 0% to 60%, except 2-way) within the first 20% of test cases. The coverage achievement of the 2-way test set significantly increases until the first 30% of the test cases. The cumulative coverage achievement of each test set increases with their corresponding interaction strength. The cumulative coverage achievement of the 2-way test set is the lowest (65.83%, 58.10%, 56.39%), while the 5-way test set achieves the highest (76.25%, 77.37%, 72.22%) coverage. Applying similar analysis for TCAS to PEL, three observation points, A, B, and C, are made at 65, 264, and 851. The details are shown in Table 9.

Table 9. Coverage Achievement at Point A, B, and C for PEL

	2-way	3-way	4-way	5-way	
A	65.83%	63.54%	63.54%	46.25%	Statement
	58.10%	58.10%	59.50%	42.74%	Branch
	56.39%	53.89%	55.56%	40.28%	MC/DC
B	N/A	70.00%	65.21%	64.58%	Statement
		66.20%	63.97%	62.85%	Branch
		63.33%	59.17%	58.33%	MC/DC
C	N/A	N/A	71.67%	66.88%	Statement
			72.91%	67.32%	Branch
			69.72%	61.39%	MC/DC

The differences of all three coverage achievements of 2-way, 3-way, and 4-way are not significant at point A, except the 5-way test set is much lower than the other three. At point B, the coverage achievements of the 3-way test set are higher than 4-way and 5-way test sets. Surprisingly, the 5-way test set performs the worst at all observation points.

B. Fault Detection Stregnth

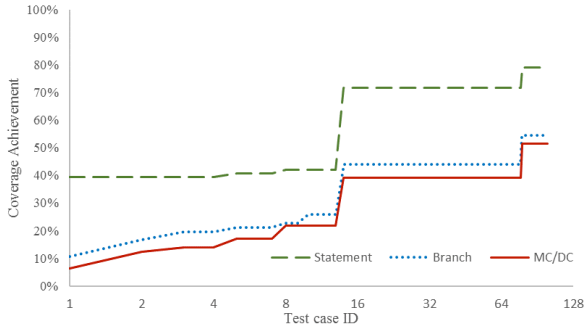
In this section, we examine the ratio (number of detected faults divided by the number of all faults) to evaluate the fault detection strength of test sets generated using CT. The ratio is denoted by fault detection rate (FDR). Table 10 and Table 11 show the results for TCAS and PEL, respectively.

Table 10. FDR of Tests of TCAS

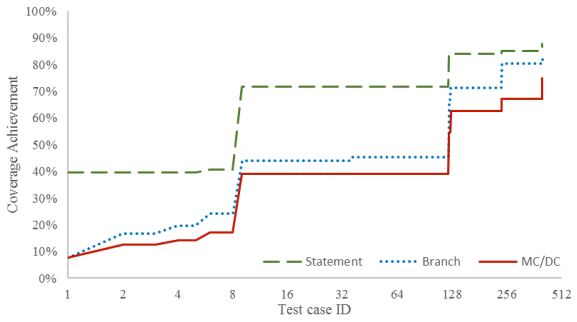
TCAS	2-way	3-way	4-way	5-way
Total number of faults detected	7	15	34	35
FDR	17.95%	38.46%	87.18%	89.74%

Table 11. FDR of Tests of PEL

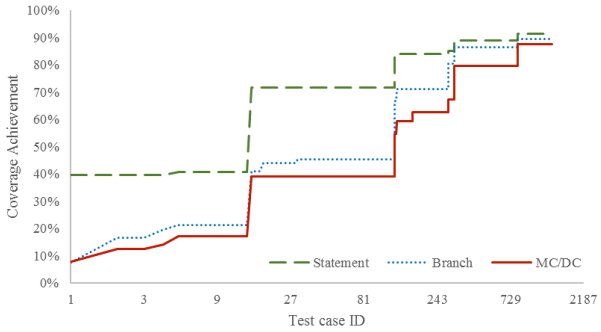
PEL	2-way	3-way	4-way	5-way
Total number of faults detected	2	5	6	9
FDR	20%	50%	60%	90%



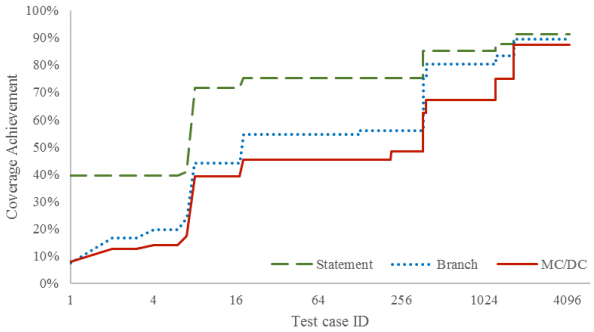
(a) 2-way Test Set



(b) 3-way Test Set

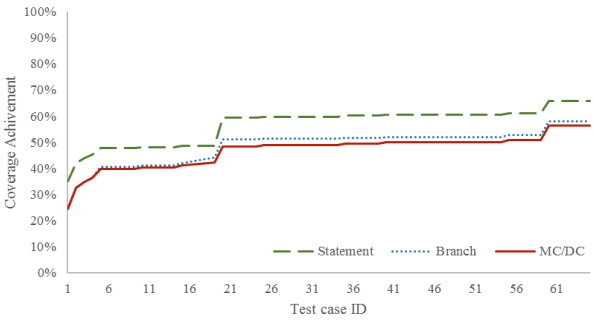


(c) 4-way Test Set

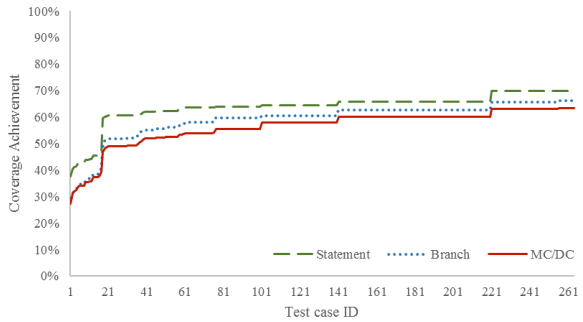


(d) 5-way Test Set

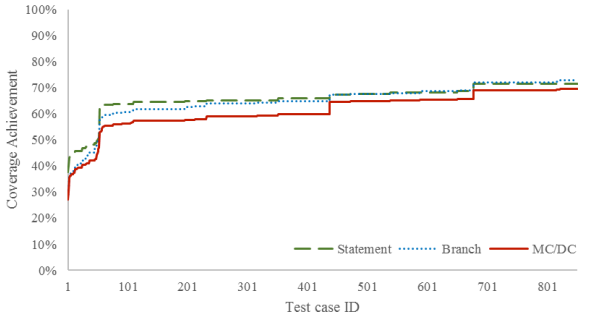
Figure 2. Coverage Achievements of Test Sets for TCAS



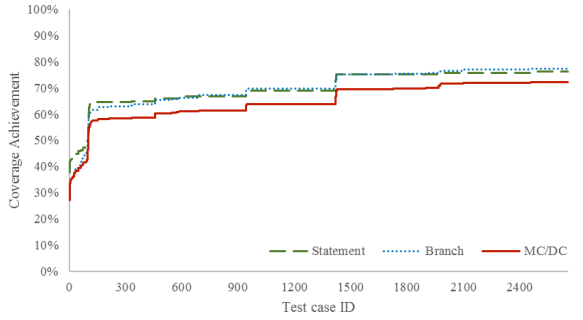
(a) 2-way Test Set



(b) 3-way Test Set



(c) 4-way Test Set



(d) 5-way Test Set

Figure 3. Coverage Achievements of Test Sets for PEL

In general, increasing interaction strength significantly improves the FDR of a test set generated by CT. However, the improvement from the 4-way to 5-way is not significant for TCAS.

In addition, we also investigate the number of test cases in each test set that detects each fault for TCAS and PEL, shown in Table 12 and Table 13, respectively.

Table 12. Number of Failed Test Cases for TCAS

	2-way	3-way	4-way	5-way
v1	0	0	1	5
v2	0	0	6	2
v3	0	1	2	12
v4	0	0	4	7
v5	0	3	25	62
v6	0	3	12	39
v7	0	0	1	6
v8	0	0	0	0
v9	0	0	0	3
v10	0	5	15	60
v11	0	5	16	74
v12	2	10	47	149
v13	0	2	8	29
v14	0	1	12	20
v15	0	3	25	62
v16	0	0	1	0
v17	0	0	1	6
v18	0	0	1	0
v19	0	0	0	1
v20	0	0	2	2
v21	0	0	1	3
v22	0	0	0	0
v23	0	0	0	2
v24	1	0	1	2
v25	0	0	2	2
v26	0	2	8	29
v28	1	0	2	7
v29	1	0	1	2
v30	0	0	1	5
v31	0	5	15	60
v32	1	6	12	53
v33	0	1	4	11
v34	3	11	73	236
v35	1	0	2	7
v36	0	0	4	6
v37	0	1	8	13
v38	0	0	3	3
v40	0	0	4	6
v41	0	0	4	7

Similar to the result of FDR, in most cases, the higher the interaction strength a test set has, the more the test cases can detect the fault. However, for v2 of TCAS, from 4-way to 5-way, the number of test cases that can detect the fault drops from six to two. The same observation can be also found in v5 of PEL from 3-way to 4-way. For v16, v18, v24, v28, v29, and v35 of TCAS, we observe that test sets with higher interaction strength cannot detect the faults, while test sets with lower interaction strength can detect the faults. By examining the test cases and their execution trace, we find out that the faults of these versions can only be detected by some special combinations of more than six input parameters. Since a test set with a lower interaction strength generated by IPOG (e.g., A) might not be a subset of a generated test set with higher interaction strength (e.g., B). Therefore, some input combinations in A, which detect the bugs, might not be included in B.

Table 13. Number of Failed Test Cases for PEL

	2-way	3-way	4-way	5-way
v1	0	1	5	64
v2	0	4	14	59
v3	0	0	0	0
v4	0	0	0	1
v5	0	10	7	46
v6	0	0	1	1
v7	1	1	18	44
v8	32	129	426	1335
v9	0	0	0	1
v10	0	0	0	1

V. Conclusion and Feature Work

In this paper, we conduct the experiment on two real-life programs with complex control flows to investigate the effectiveness and efficiency of applying combinatorial testing in order to generate tests to achieve high MC/DC. By analyzing the experiment results, the three research questions in Section I can be answered as follows:

- Can CT generate test cases to achieve high MC/DC?

  Our experiment results show that test sets generated using CT can achieve high MC/DC. In general, the higher interaction strength a CT test set has, the higher MC/DC it can achieve.

- How to use CT to improve coverage achievements (statement, branch, and MC/DC) in an efficient way?

  According to the experiment results, choosing the appropriate interaction strength is very critical to achieving high efficiency. The intuitive assumption that, "the higher interaction strength a test set has, the higher coverage it can achieve," holds in most cases. However, our experiment results also show that the cumulative coverage achievements of the three criteria of the 5-way test set are not significantly improved comparing with the 4-way test set. In general, the coverage achievements of the 5-way test set increase more slowly than the 2-way, 3-way, and 4-way test sets. Because the 2-way test set is not effective in achieving cumulative coverage achievements, we recommend using the 3-way or 4-way test sets in practice.

- Can tests generated using CT achieve high fault detection strength?

  In most cases, a test set (e.g., A) with a higher interaction strength detects more faults than a test set (e.g., B) with a lower interaction strength. In some cases, we also notice that there are some faults that cannot be detected by A but can be detected by B. Therefore, we suggest that the practitioners carefully examine the coverage report after testing using CT to determine whether the SUT is tested adequately to avoid such cases.

In the future, we are planning to apply CT to more programs, and also to investigate the impact of the input model on coverage achievements, as well as the fault detection strength.

ACKNOWLEDGMENT

Disclaimer: Any mention of commercial products in this paper is for information only; it does not imply recommendation or endorsement by NIST.

REFERENCES

[1] American Airlines Flight 965: Crash on the Mountain (http://aviationknowledge.wikidot.com/asi:american-airlines-flight-965:crash-on-the-mountain; accessed Nov 8, 2016)

[2] RTCA, DO-178B: Software considerations in airborne systems and equipment certification. Washington, RTCA, Inc., December 1992

[3] RTCA, DO-178C: Software considerations in airborne systems and equipment certification. Washington, RTCA, Inc., December 2011

[4] A. Dupuy, and L. Nancy , "An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software" in *Proceedings of the IEEE 19th Digital Avionics Systems Conference,* vol. 1, pp. 1B6-1, Philadelphia, PA, USA, October, 2000

[5] A. L. White. "Comments on modified condition/decision coverage for software testing [of flight control software]." *In Proceedings of the IEEE Aerospace Conference*, vol. 6, pp. 2821-2827. Big Sky, USA, March, 2001

[6] C. Cadar, D. Dunbar, and D. Engler, "KLEE: Unassisted and Automatic Generation of High-Coverage tests for Complex Systems Programs," in *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*, pp. 209-224, San Diego, USA, January 2008

[7] S.R. Ganov, C. Killmar, S. Khurshid, and D.E., Perry, "Test generation for graphical user interfaces based on symbolic execution", in *Proceedings of the 3rd IEEE/ACM International Workshop on Automation of Software Test*, pp. 33-40, Leipzig, Germany, May 2008.

[8] M. Papadakis and N. Malevris, "Automatic Mutation Test Case Generation Via Dynamic Symbolic Execution", in *Proceedings of the 21st International Symposium on Software Reliability Engineering*, pp. 121-130, San Jose, CA, USA, November 2010

[9] S. Anand, E. Burke, T. Chen, J. Clark, and M. Cohen, "An orchestrated survey of methodologies for automated software test case generation", *Journal of Systems and Software* 86(8):1978-2001, 2013

[10] J. Bozic, B. Garn, I. Kapsalis, D. E. Simos, Severin Winkler and F. Wotawa, "Attack Pattern-Based Combinatorial Testing with Constraints for Web Security Testing," in *Proceedings of the 2015 IEEE International Conference on Software Quality, Reliability and Security* (QRS), pp. 207-212, Vancouver, Canada, August 2015

[11] J. D. Hagar, D. R. Kuhn, and R. N. Kacker, and T. L. Wissink, "Introducing Combinatorial Testing in a Large Organization: Pilot Project Experience Report," in *Proceedings of the Seventh IEEE International Conference on Software, Testing, Verification and Validation Workshops* (ICST Workshops), pp. 153, 2014

[12] D. R. Kuhn and R. N. Kacker, "Practical Combinatorial (t-way) Methods for Detecting Complex Faults in Regression Testing," in *Proceedings of the IEEE 27th International Conference on Software Maintenance* (ICSM), pp. 599, September, 2011

[13] M. Palacios, J. Garcia-Fanjul, J. Tuya, and G. Spanoudakis, "Automatic Test Case Generation for WS-Agreements using Combinatorial Testing," *Computer Standards & Interfaces*, vol. 38, pp. 84-100, 2015.

[14] M. Forbes, J. Lawrence, Y. Lei, R.N. Kacker, and D.R. Kuhn Refining the In-Parameter-Order Strategy for Constructing Covering Arrays, NIST Journal of Research, 113(5):287-297, Sept./Oct., 2008

[15] 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, pp. 125-148, 2008

[16] D. R. Kuhn, I. D. Mendoza, R. N. Kacker, and Y. Lei, "Combinatorial Coverage Measurement Concepts and Applications," in *Proceedings of* the Sixth IEEE International Conference on Software, Testing, Verification and Validation Workshops (ICST Workshops), pp. 352-361, 2013

[17] D. R. Kuhn and V. Okun, "Pseudo-exhausive Testing for Software," in Proceedings of IEEE/NASA Software Engineering Workshop, pp.153-158, Columbia, USA, April 2006

[18] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, "An evaluation of similarity coefficients for software fault localization." in *Proceedings of 12th Pacific Rim International Symposium on Dependable Computing*, pp. 39-46, Riverside, CA, USA, December, 2006

[19] X. Li, W. E. Wong, R. Gao, L. Hu, and S. Hosono, "Genetic Algorithm-based Test Generation for Software Product Line with the Integration of Fault Localization Techniques." Empirical Software Engineering: pp. 1-51. 2017

[20] ACTS, a combinatorial test genertion tool, http://csrc.nist.gov/groups/SNS/acts/

[21] The Software Infrastructure Repository (retrieved October 2008) (http://sir.unl.edu/portal/index.html)

[22] John J. Chilenski, "An investigation of three forms of the modified condition decision coverage (MCDC) criterion," Tech. Rep. DOT/FAA/AR-01/18, Federal Aviation Administration, US-Department of Transportation, Washington, DC, April 2001

[23] M. Palacios, J. Garcia-Fanjul, J. Tuya, and G. Spanoudakis, "Automatic Test Case Generation for WS-Agreements using Combinatorial Testing," *Computer Standards & Interfaces*, vol. 38, pp. 84-100, 2015.

[24] X. Li, R. Gao, W. E. Wong, C. Yang, and D. Li, "Applying Combinatorial Testing in Industrial Settings", in *Proceedings of the 2016 IEEE International Conference on Software Quality, Reliability and Security* (QRS), pp. 53-60, Vienna, Austria, October, 2016

[25] NIST Report, "Software Errors Cost U.S. Economy $59.5 Billion Annually", NIST Planning Report 02-3, May 2002J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, pp.68-73, 2002