

Combinatorial Test Case Prioritization based on Incremental Combinatorial Coverage Strength

Ziyuan Wang* and Feiyan She

School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing, 210023, China

Abstract

A combinatorial test case prioritization technique based on incremental combinatorial coverage strength is proposed in this paper. Such an efficient technique prioritizes test cases in an existing high-strength combinatorial test suite, to form an ordered test case sequence. The output prioritized combinatorial test suite could cover combinations of parametric values, where their strengths are mixed from 2 or more, quicker than non-prioritized combinatorial test suite. Theoretical analysis shows that the proposed technique consumes less cost than existing combinatorial test case prioritization technique for the same type of scenarios. Experimental results also show that compared to initial combinatorial test suites without prioritization, prioritized combinatorial test suites yield a higher speed of covering combinations of parametric values. Compared to combinatorial test suites generated by an existing incremental adaptive combinatorial testing strategy, prioritized combinatorial test suites 1) require less test cases to achieve final combinatorial coverage, and 2) yield a higher speed of covering high-strength combinations of parametric values.

Keywords: software testing; combinatorial testing; test case prioritization; coverage strength; average percent of combinatorial coverage

(Submitted on March 16, 2018; Revised on April 26, 2018; Accepted on May 27, 2018)

© 2018 Totem Publisher, Inc. All rights reserved.

1. Introduction

Combinatorial testing, which provides a tradeoff between the cost of testing and the degree of combinatorial coverage, has been widely utilized in highly-configurable systems for its efficiency and effectiveness, e.g., a τ -way combinatorial test suite could cover all τ -tuple combinations of parametric values [16]. In traditional process of combinatorial testing, a combinatorial test suite is considered as an entire. That means, all test cases in a combinatorial test suite with a given combinatorial coverage strength should be running fairly and completely, in order to achieve the given combinatorial coverage criteria. However, in many applications where combinatorial testing is needed (e.g. regression testing), the entire combinatorial test suite is not run because of testing resource constraints. To improve the efficiency of testing, combinatorial test case prioritization, which prioritizes combinatorial test cases to cover combinations of parametric values as rapidly as possible, is required to extend the traditional combinatorial testing process.

A combinatorial test case prioritization technique, which prioritizes test cases in an existing τ -way combinatorial test suite to form an ordered test case sequence, was proposed by Bryce and Memon [3, 4]. The output of such technique is a prioritized τ -way combinatorial test suite, which covers τ -tuple combinations of parametric values more rapidly than initial non-prioritized combinatorial test suite. There are two limitations of such an existing technique: 1) Only τ -tuple combinations of parametric values are taken into consideration; however, there are also missions of τ -way combinatorial test suite that cover combinations with strength less than τ ; 2) It may be high time complexity to compute test cases' coverage abilities for τ -tuple combinations of parametric values in high-strength combinatorial testing. There is an incremental adaptive combinatorial testing strategy proposed by Fouche and Cohen et al [9]. Such incremental strategy could be utilized in combinatorial test case prioritization to avoid above limitations.

In this paper, a combinatorial test case prioritization technique based on incremental combinatorial coverage strength is

* Corresponding author.

E-mail address: wangziyuan@njupt.edu.cn

presented. The proposed technique could prioritize existing high-strength τ -way combinatorial test case efficiently. The prioritized combinatorial test suite could cover combinations of parametric values, where their strengths are mixed from 2 to τ , more rapidly than initial non-prioritized combinatorial test suite. Theoretical analysis shows that the proposed technique consumes less cost than existing combinatorial test case prioritization technique for the same type of scenarios. Experimental results also show that compared to initial non-prioritized combinatorial test suites, prioritized combinatorial test suites yield a higher speed of covering combinations of parametric values. Compared to combinatorial test suites generated by an existing incremental adaptive combinatorial testing strategy, prioritized combinatorial test suites 1) require less test cases to achieve final combinatorial coverage, and 2) yield a higher speed of covering high-strength combinations of parametric values.

The rest of this paper is organized as follows. The 2nd section introduces some preliminaries. The 3rd section introduces the motivation of the combinatorial test case prioritization technique based on incremental combinatorial coverage strength. The 4th describes the algorithm in detail. The 5th section gives experimental results and the 6th section discusses the results. Related works and conclusion remarks are addressed at the end.

2. Preliminaries

Some preliminaries about combinatorial testing and test case prioritization are presented in this section.

2.1. Combinatorial testing

Assume that there is a testing subject with n input variables $F=\{f_1, f_2, \dots, f_n\}$, where each input variable has a_i input values to form a value set $V_i=\{1, 2, \dots, a_i\}$ for $i=1, 2, \dots, n$. Let $a=\max_{1 \leq i \leq n}\{a_i\}$, if the cardinalities of value sets of all input variables are uniform ($a_1=a_2=\dots=a_n=a$), the testing subject could be called a fixed-level system with an input model $M=\{a^n\}$ or an a -level system. Otherwise, the testing subject could be called a mixed-level system with an input model $M=\{a_1 \times a_2 \times \dots \times a_n\}$.

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 a symbols at least once. Here τ is called the combinatorial coverage strength of such a fixed-level covering array.

Definition 2. A τ -way mixed-level covering array $CA(m; \tau, (a_1, a_2, \dots, a_n))$ is an $m \times n$ array on totally $a=\max_{1 \leq i \leq n}\{a_i\}$ symbols with the properties that, (1) the i -th column contains only elements from value set $V_i=\{1, 2, \dots, a_i\}$ for $i=1, 2, \dots, n$; and (2) each $m \times \tau$ sub-array contains all possible τ -tuples from corresponding value sets at least once. Here τ is called the combinatorial coverage strength of such a mixed-level covering array.

Both τ -way fixed-level covering array and τ -way mixed-level covering array could be called a τ -way covering array or a covering array with strength τ . A τ -way covering array must cover all the τ -tuples in the following set [21]:

$$CombSet(\tau)=\bigcup_{r \in R} CombSet_r(\tau) \quad (1)$$

Where

$$R=\{\{f_{i,1}, f_{i,2}, \dots, f_{i,\tau}\} \mid f_{i,1}, f_{i,2}, \dots, f_{i,\tau} \in F\}$$

$$CombSet_r(\tau)=\{(v_{i,1}, v_{i,2}, \dots, v_{i,\tau}) \mid v_{i,1} \in V_{i,1}, v_{i,2} \in V_{i,2}, \dots, v_{i,\tau} \in V_{i,\tau}\}$$

For a given testing subject, a test suite could be obtained from τ -way covering array by mapping each row of covering array, which is a n -tuple (v_1, v_2, \dots, v_n) where $v_1 \in V_1, v_2 \in V_2, \dots, v_n \in V_n$, to a test cases for such a testing subject. Such software testing approach that designs and runs a τ -way covering array as a test suite is called the τ -way combinatorial testing. So, we usually assume that combinatorial test suite and covering array are equivalent.

Since small size of test suite leads to cheaper software testing process, many algorithms have been proposed to generate combinatorial test suite with small size. All these algorithms could be mainly classified into orthogonal array-based method [15], recursive construction method [6], one-test-at-a-time strategy [2], in-parameter-order strategy [14], etc.

2.2. Test case prioritization

Test case prioritization, which seeks to schedule test cases in an order that increases the effectiveness at meeting some given

performance goal, is an effective testing optimization technique. It provides a priority for each test case, and then selects and executes test cases according to high to low priority order. The test case with higher priority will be executed as early as possible. A classic test case prioritization problem can be defined as:

Definition 3. For an initial non-prioritized test suite T_{init} , the test case prioritization aims to find a best prioritized test suite $T \in PT$ such that:

$$(\forall T') (T' \in PT) (T \neq T') [f(T) \geq f(T')] \quad (2)$$

Where PT is the set of all possible permutations of T_{init} (PT collects all possible prioritized test suites that contain all test cases in T_{init}). f is an objective function; if applied to any such ordering, it yields an award value for that ordering [7, 17].

There are several possible objective functions for test case prioritization problem. People usually restrict their attention to the speed of fault detection, since fault detection should be one of final goals of software testing. Therefore, an objective function called average percent of faults detected (*APFD*), was proposed by G. Rothermel et al. as a metric to evaluate the speed of fault detection [7, 17]. Consider test suite $T = \{T_1, T_2, \dots, T_m\}$ and fault set $FaultSet = \{fault_1, fault_2, \dots, fault_k\}$ of testing subject. If tf_i is denoted as the index of the first test case that detects the $fault_i$, the *APFD* for T is:

$$APFD(T) = 1 - \frac{tf_1 + tf_2 + \dots + tf_k}{k \times m} + \frac{1}{2m} \quad (3)$$

A prioritized test suite with higher *APFD* value has better fault detection efficiency than that with lower *APFD* values. The range of *APFD* value should be (0, 1) [24].

In order to increase *APFD* value of prioritized test suite, the intuitive approach is to prioritize test cases according to the relationship between test suite T and fault set *FaultSet*. However, it is impossible to obtain such a relationship before executing test cases. So, it is more practical to prioritize test cases using some other types of information. E.g., there are test case prioritization techniques based on source code and test history [23], varying fault severities and test costs [8], basic block coverage [18], time budget [19], program slicing [13], etc. In this paper, we will prioritize combinatorial test cases based on test cases' coverage abilities for combinations of parametric values.

3. Motivation

A combinatorial test case prioritization technique (CTPri), which prioritizes test cases in an existing τ -way combinatorial test suite to form an ordered test case sequence, was proposed by Bryce and Memon [3, 4]. It orders test cases according to their coverage abilities for τ -tuple combinations of parametric values. The test cases that cover more uncovered τ -tuple combinations will be selected earlier than others. So, the output prioritized τ -way combinatorial test suite covers τ -tuple combinations more rapidly than initial non-prioritized combinatorial test suite. There are two limitations of such an existing technique: 1) Only τ -tuple combinations of parametric values are taken into consideration; however, there are also missions of τ -way combinatorial test suite that cover combinations with strength less than τ ; 2) It may be high time complexity to compute test cases' coverage abilities for τ -tuple combinations of parametric values in high-strength combinatorial testing.

There is an incremental adaptive combinatorial testing strategy proposed by Fouche and Cohen et al [9]. Such strategy generates a high-strength combinatorial test suite by an iterative process. First, a 2-way combinatorial test suite is generated. It could be utilized as seed test cases to generate a 3-way combinatorial test suite, which can be utilized as seed test cases to generate a 4-way combinatorial test suite. Incremental adaptive combinatorial testing strategy repeats such a process iteratively until the coverage strength meets a given upper bound. In such strategy, the concrete combinatorial test generation algorithm in each step may be any possible combinatorial test generation algorithm that supports seed test cases, such as algorithms in one-test-at-a-time framework. For a given upper bound of combinatorial coverage strength $max_strength$ ($max_strength > 2$), the process of incremental adaptive combinatorial testing strategy could be divided into $max_strength - 1$ steps. In the i -th step, where $i = 1, 2, \dots, max_strength - 1$, a $(i+1)$ -way combinatorial test suite will be generated to cover all $(i+1)$ -tuples.

To avoid limitations of CTPri algorithm, the incremental strategy could be utilized to prioritize test cases in high-strength combinatorial test suite. E.g., considering a fixed-level system with input model $M = \{2^4\}$, a 4-way combinatorial test suite with 16 test cases could be found in Table 1. Because the most popular combinatorial testing is 2-way

combinatorial testing (sometimes called pairwise testing), we start test case prioritization process for 2-tuple combinations of parametric values in the first step. Test cases with index 1, 8, 11, 14, 2, 5 will be selected one by one. The numbers of newly covered 2-tuple combinations by them are 6, 6, 5, 5, 1, 1 respectively. Such an order could cover all 24 2-tuple combinations with the highest speed. Then, select test case for rest 3-tuple combinations in the second step to obtain a sequence that cover rest 3-tuple combinations with the highest speed. Finally, select test case for rest 4-tuple combinations. There will be a prioritized test suite after the third step.

Table 1. 4-way combinatorial test suite for input model $M=\{2^4\}$

No.	Input1	Input2	Input3	Input4	No.	Input1	Input2	Input3	Input4
1	1	1	1	1	9	2	1	1	1
2	1	1	1	2	10	2	1	1	2
3	1	1	2	1	11	2	1	2	1
4	1	1	2	2	12	2	1	2	2
5	1	2	1	1	13	2	2	1	1
6	1	2	1	2	14	2	2	1	2
7	1	2	2	1	15	2	2	2	1
8	1	2	2	2	16	2	2	2	2

That means, for a combinatorial test suite with strength $max_strength$ ($max_strength > 2$), there are $max_strength-1$ steps in combinatorial test case prioritization. In the i -th step, where $i=1, 2, \dots, max_strength-1$, test cases will be selected to cover $(i+1)$ -tuple combinations of parametric values that have not been covered by selected test cases.

4. Algorithm

The whole process of combinatorial test case prioritization based on incremental combinatorial coverage strength could be divided into $max_strength-1$ steps for an initial non-prioritized $max_strength$ -way combinatorial test suite:

- (1) In the first step, the CTPri algorithm could be utilized to select test cases to cover 2-tuple combinations of parametric values until all 2-tuple combinations have been covered. Each selected test case should cover the most number of uncovered 2-tuple combinations possible. A partial test case sequence where all 2-tuple combinations are covered can be obtained in this step. Note that some 3-tuple combinations are covered by such partial test sequence too..
- (2) In the second step, test cases will be selected to cover rest 3-tuple combinations until all 3-tuple combinations have been covered. Each selected test case should cover the most number of uncovered 3-tuple combinations possible. An extended partial test case sequence where all 3-tuple combinations are covered can be obtained in this step. Note that some 4-tuple combinations are covered too.
- (3) Repeat until all $max_strength$ -tuple combinations have been covered in the $(max_strength-1)$ -th step. A final prioritized combinatorial test suite could be obtained. If there is not any redundant test case in initial test suite, all $max_strength$ -tuple combinations are covered, which means that all test cases in initial test suite are selected.

The pseudo-code of above process can be found in Figure 1.

Input: initial test suite T_{init}
Output: test case sequence T

```

1 Initialize  $T[1 \dots |T_{init}|][1 \dots n]$ 
2  $index := 1$ 
3 Select a test case  $test$  from  $T_{init}$  randomly,  $T[index] := test$ 
4 for  $\tau := 2$  to  $max\_strength$ 
5   Initialize  $CombSet(\tau)$  //Generate  $CombSet(\tau)$ , a set of  $\tau$ -tuples for input model
6   for  $i := 1$  to  $index$ 
7     Modify  $CombSet(\tau)$ , remove tuples that covered by  $T[i]$ 
8   end for
9   while ( $CombSet(\tau) \neq \emptyset$ )
10     $Max\_index := -1$ ;  $Max\_Count := -1$ 
11    for  $i := 1$  to  $|T_{init}|$ 
12       $Count := CoverNum(T[i], \tau)$  //Count the number of tuples that covered by  $T[i]$  in set  $CombSet(\tau)$ 
13      if ( $Max\_Count < Count$ ) then
14        { $Max\_index := i$ ;  $Max\_Count := Count$  }
15      else
16        if ( $Max\_Count == Count$  AND Random[0,1)<0.5) then //Tie-break randomly
17          { $Max\_index := i$ ;  $Max\_Count := Count$  }
18        end if
19      end if

```

```

20  end for
21  T[++index] := T[Max_index]
22  Modify CombSet( $\tau$ ), remove tuples that covered by T[Max_index]
23  end while
24  end for

```

Figure 1. Combinatorial test case prioritization based on incremental combinatorial coverage strength (InCTPri)

By assuming that $\tau = \text{max_strength}$, the time complexity of CTPri algorithm should be $O(m^2 \times C_n^\tau \times \log(a^\tau))$. The detail analysis process could be found in [22]. To analyze the time complexity of proposed algorithm, we assume that in the i -th ($i=1, 2, \dots, \text{max_strength}-1$) step, m_i test cases are selected. That is, $m = m_1 + m_2 + \dots + m_{(\text{max_strength}-1)}$. In the i -th step, when selecting the j -th test case in prioritized test suite, the number of test cases that need to calculate the number of newly covered tuples is $m-j+1$. That means the total times of calculating the number of newly covered tuples in i -th step is $O(m \times m_i)$. Therefore, the time complexity of proposed algorithm is $O(\sum_i m \times m_i \times C_n^{i+1} \times \log(a^{i+1}))$. It is better than that of CTPri algorithm proposed by Bryce and Memon.

Theorem 1. If there is redundant τ -tuple combination in a combinatorial test suite, there must be redundant $(\tau-1)$ -tuple combinations in such a combinatorial test suite.

Proof. Suppose two test cases test_1 and test_2 in a combinatorial test suite T cover the same τ -tuple $(v_{i,1}, v_{i,2}, \dots, v_{i,\tau})$ where $v_{i,1} \in V_{i,1}, v_{i,2} \in V_{i,2}, \dots, v_{i,\tau} \in V_{i,\tau}$ and $f_{i,1}, f_{i,2}, \dots, f_{i,\tau} \in F$. There are totally τ different $(\tau-1)$ -tuples $(v_{j,1}, v_{j,2}, \dots, v_{j,\tau-1})$ where $v_{j,1} \in V_{j,1}, v_{j,2} \in V_{j,2}, \dots, v_{j,\tau-1} \in V_{j,\tau-1}$ and $f_{j,1}, f_{j,2}, \dots, f_{j,\tau-1} \in \{f_{i,1}, f_{i,2}, \dots, f_{i,\tau}\}$ by picking up $\tau-1$ values from the τ -tuple. All these $(\tau-1)$ -tuples are covered by both test_1 and test_2 . It means that there are redundant $(\tau-1)$ -tuple combinations. ■

The contrapositive of such theorem tells us that if there is not any redundant $(\tau-1)$ -tuple combination in a combinatorial test suite, there will not be any redundant τ -tuple combination. Therefore, in the i -th ($i=1, 2, \dots, \text{max_strength}-1$) step of test case prioritization process, the test case that covers $(i+1)$ -tuple combinations of parametric values rapidly must cover higher-strength combinations rapidly too. In another words, the proposed algorithm can output the same results as the CTPri algorithm, but with better time performance.

5. Experiment

There are two research questions to answer in our experiments: 1) Can that proposed algorithm increase the speed of combinatorial coverage? 2) Are there advantages or disadvantages when comparing to incremental adaptive combinatorial testing strategy?

5.1. Design

We implement incremental adaptive combinatorial testing algorithm named as InCTGen, according to the description provided by Fouche and Cohen et al [9]. In order to reduce the size of the generated test suite, different types of algorithms including orthogonal array [15], recursive construction method [6], one-test-at-a-time algorithm called DDA [2], in-parameter-order algorithm called IPO [14] are utilized comprehensively in the first step of InCTGen. The best result with less test cases will be used as the first-step test suite. In subsequent steps, only DDA algorithm is utilized since it supports seed test cases. We also implement algorithm 1 named as InCTPri. The above four algorithms are utilized to generate initial combinatorial test suites. The best result with less test cases will be used as the input of InCTPri.

We select four input models that were often used previously in our experiment: $M=\{4^5\}$, $M=\{5^6\}$, $M=\{2^3 \times 3^3 \times 4^3 \times 5\}$, and $M=\{2^4 \times 6^2 \times 7^2 \times 8^2\}$. The maximum combinatorial coverage strength is set as $\text{max_strength}=4$. For each input model, a 4-way combinatorial test suite T_{Gen} is generated by InCTGen algorithm. Meanwhile, a 4-way combinatorial test suite T_{init} is generated as the input of InCTPri algorithm. The InCTPri algorithm prioritizes test cases in T_{init} to form a prioritized test suite T_{pri} . In our experiment, T_{Gen} , T_{init} , and T_{pri} will be taken into comparison.

5.2. Metrics

For test suite T_{Gen} , T_{init} , and T_{pri} , we count the number of test cases that achieve 2-way, 3-way and 4-way combinatorial coverage criterion. The speeds of combinatorial coverage for 2-way, 3-way and 4-way tuples will be taken into comparison too. APCC metric will be utilized to evaluate how rapidly a combinatorial test suite covers combinations of parametric values [22]:

$$APCC(T) = \frac{\sum_{i=1}^{m-1} \left| \bigcup_{j=1}^i tCov(T_j) \right|}{m \times |CombSet|} \quad (4)$$

Where the function $tCov(T_j)$ returns a set of all τ -tuple combinations that covered by the test case T_j . Due to the property of APCC metric, it only works when compare two test suites with the same numbers of test cases. So, if $|T_{init}| < |T_{gen}|$, $|T_{gen}| - |T_{init}|$ additional test cases should be inserted into the back of T_{pri} . And vice versa.

5.3. Results

The results of experiments are shown in Table 2. We can conclude from experiment data that:

- (1) The sizes of T_{init}/T_{pri} are always smaller than the sizes of T_{gen} . That means, InCTPri needs less test cases than InCTGen to achieve high-strength combinatorial coverage. But when covering 2-tuple and 3-tuple combinations, required test cases in T_{pri} are always more than that of T_{gen} , which means InCTGen needs less test cases than InCTPri to achieve low-strength combinatorial coverage.
- (2) For 4-tuple combinations, the APCC values of T_{pri} are always greater than that of T_{gen} . That means, InCTPri makes a more rapid combinatorial coverage for high-strength combinations. But for 2-tuple and 3-tuple combinations, the APCC values of T_{pri} are usually less than that of T_{gen} , which means InCTGen makes a more rapid combinatorial coverage for low-strength combinations.
- (3) For 2-tuple, 3-tuple, and 4-tuple combinations, APCC values of T_{pri} are always greater than that of T_{init} , which means InCTPri increases the speed of combinatorial coverage.

Table 2. Compare non-prioritized test suites and prioritized test suites obtained by InCTGen and InCTPri

(a) $M=\{4^5\}$, $Max_Strength=4$ The input of InCTPri is OA Use OA and DA-RO algorithm in InCTGen					(b) $M=\{5^6\}$, $Max_Strength=4$ The input of InCTPri is OA Use OA and DA-RO algorithm in InCTGen				
		2-way	3-way	4-way			2-way	3-way	4-way
InCTGen	# test cases	16	64	362	InCTGen	# test cases	34	215	1055
	APCC _{τ}	0.9668	0.8730	0.4808		APCC _{τ}	0.9807	0.8936	0.4604
InCTPri	# test cases	24	108	256	InCTPri	# test cases	41	207	725
	APCC _{τ}	0.9641	0.8497	0.4980		APCC _{τ}	0.9609	0.8584	0.4993
Initial	APCC _{τ}	0.9556	0.7944	0.4631	Initial	APCC _{τ}	0.9257	0.7833	0.4483

(c) $M=\{2^3 \times 3^3 \times 4^3 \times 5\}$, $Max_Strength=4$ The input of InCTPri is generated by DA-RO algorithm Use DA-RO algorithm in InCTGen					(d) $M=\{2^4 \times 6^2 \times 7^2 \times 8^2\}$, $Max_Strength=4$ The input of InCTPri is generated by DA-FO algorithm Use DA-RO algorithm in InCTGen				
		2-way	3-way	4-way			2-way	3-way	4-way
InCTGen	# test cases	25	112	421	InCTGen	# test cases	36	140	371
	APCC _{τ}	0.9816	0.9357	0.7390		APCC _{τ}	0.9758	0.9220	0.7565
InCTPri	# test cases	27	127	412	InCTPri	# test cases	39	147	363
	APCC _{τ}	0.9391	0.9345	0.7902		APCC _{τ}	0.9746	0.9272	0.8187
Initial	APCC _{τ}	0.9254	0.9169	0.6502	Initial	APCC _{τ}	0.9387	0.9015	0.6833

Experimental results suggest that: 1) It is always useful to adopt test case prioritization in combinatorial testing. 2) The strategy that has a higher efficiency of combinatorial coverage is needed when the testing resource is limited. So, InCTGen should have a higher priority in this circumstance. 3) If there is no risk that testing may be terminated exceptionally, higher efficiency of combinatorial coverage is not important. The strategy that requires a smaller final test suite is better and that InCTPri should have a higher priority.

6. Related works

Works about combinatorial test case prioritization can be addressed in two strategies: 1) prioritize test cases in an existing combinatorial test suite, and 2) incorporate test prioritization into combinatorial test generation.

Bryce et al. first attempted to prioritize test cases in an existing combinatorial test suite [3, 4]. Wang et al. proposed to poetize combinatorial test cases according to the weights of combinations and the costs of test execution [22]. He also

proposed to prioritize combinatorial test cases for incremental combinatorial coverage strength in his Ph.D dissertation in Chinese [20]. This paper could be considered an extended version of that. Huang et al. did the same work [12]. The technique based on adaptive random strategy [10] and the technique for aggregate-strength [11] were proposed in recent years.

In the field of prioritized combinatorial test case generation, Bryce et al. provided the pioneer work too [1]. They extend a well-known combinatorial test generation algorithm DDA for the scenario that weights are taken into consideration. Chen et al. proposed to generate prioritized combinatorial test suites with ant colony algorithm [5].

7. Conclusions

This paper proposes a combinatorial test case prioritization technique based on incremental combinatorial coverage strength. It provides an iterate process to prioritize test cases in an existing high-strength combinatorial test suite, according to the combinatorial coverage abilities of test cases for combinations of parametric values with multiple strengths. The proposed technique could make combinatorial test suite cover combinations more rapidly. Though there is an existing combinatorial test case prioritization technique for the same type of scenarios, theoretical analysis shows that the proposed technique is less expensive. Experimental results indicate that compared to initial combinatorial test suites without prioritization, prioritized combinatorial test suites yield a higher speed of covering combinations of parametric values. Compared to test suites generated by the existing incremental adaptive combinatorial testing strategy, prioritized combinatorial test suites 1) require less test cases to achieve final combinatorial coverage, and 2) yield a higher speed of covering high-strength combinations of parametric values. It suggests that proposed technique may be suitable for scenarios where there are lower risks that testing may be terminated exceptionally.

In future works, more experiments should be conducted to examine the effectiveness of combinatorial test prioritization techniques. It is meaningful since there have been some existing works about combinatorial test case prioritization. The base of these works is to construct more benchmark data set for the studies of combinatorial testing.

Acknowledgements

Works described in this paper were supported by the National Natural Science Foundation of China (61772259).

References

1. R. C. Bryce, C. J. Colbourn, "Test Prioritization for Pairwise Interaction Coverage," In *Proceedings of 1st International Workshop on Advances in Model-Based Testing*, pp. 1-7, St Louis, Missouri, USA, May 15-21, 2005.
2. R. C. Bryce, C. J. Colbourn. A Density-Based Greedy Algorithm for Higher Strength Covering Arrays. *Software Testing, Verification and Reliability*, 2009, 19(1): 37-53.
3. R. C. Bryce, A. M. Memon. "Test Suite Prioritization by Interaction Coverage". In *Proceedings of Workshop on Domain-Specific Approaches to Software Test Automation (DoSTA2007)*, Dubrovnik, Croatia, September 4, 2007.
4. R. C. Bryce, S. Sampath, A. M. Memon. "Developing a Single Model and Test Prioritization Strategies for Event-Driven Software". *IEEE Transactions on Software Engineering*, 2011, 37(1): 48-64.
5. Xiang Chen, Qing Gu, Xin Zhang, Daoxu Chen. Building Prioritized Pairwise Interaction Test Suites with Ant Colony. In *Proceedings of the 9th International Conference on Quality Software (QSIC2009)*, August 24-25, 2009: 347-352.
6. C. J. Colbourn, S. S. Martirosyan, T. V. Trung, R. A. Walker II. Roux-Type Constructions for Covering Arrays of Strengths Three and Four. *Designs, Codes and Cryptography*, 2006, 41(1): 33-57.
7. S. Elbaum, A. Malishevsky, G. Rothermel, "Prioritizing Test Cases for Regression Testing," in *Proceedings of International Symposium on Software Testing and Analysis (ISSTA2000)*: 102-112.
8. S. Elbaum, A. Malishevsky, G. Rothermel. Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE2001)*: 329-338.
9. S. Fouché, M. B. Cohen, A. Poter, "Towards Incremental Adaptive Covering Arrays," in *Proceedings of 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE2007)*, pp. 557-560, Dubrovnik, Croatia, September 3-7, 2007.
10. R. Huang, J. Chen, Z. Li, R. Wang, Y. Lu. Adaptive random prioritization for interaction test suites. In: *Proceedings of the 29th Symposium on Applied Computing (SAC'14)*: 1058-1063.
11. R. Huang, J. Chen, D. Towey, A. T. S. Chan, Y. Lu. "Aggregate-strength interaction test suite prioritization". *Journal of Systems & Software*, 2015, 99(C):36-51.
12. Rubing Huang, Xiaodong Xie, Dave Towey, Tsong Yueh Chen, Yansheng Lu, Jinfu Chen. "Prioritization of Combinatorial Test Cases by Incremental Interaction Coverage". *International Journal of Software Engineering and Knowledge Engineering*, 2013.
13. D. Jeffrey, N. Gupta. Test Case Prioritization Using Relevant Slices. In *Proceedings of the 29th International Computer*

- Software and Applications Conference (COMPSAC2006)*, Chicago, USA. pp. 411-420.
14. Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, J. Lawrence. IPOG IPOG-D: Efficient Test Generation for Multi-Way Combinatorial Testing. *Software Testing, Verification and Reliability*, 2008, 18(3): 125-148.
 15. R. Mandl. Orthogonal Latin Squares: An Application of Experimental Design to Compiler Testing. In *Communications of the ACM*, 1985, 28(10): 1054-1058.
 16. Changhai Nie, Hareton Leung. "A survey of combinatorial testing". *ACM Computing Surveys (CSUR)*, 2011, 43(2):1-29.x
 17. G. Rothermel, R. H. Untch, C. Y. Chu, M. J. Harrold, "Prioritizing Test Cases for Regression Testing," *IEEE Transactions on Software Engineering*, 2001, 27(10): 929-948.
 18. A. Srivastava, J. Thiagrajan. Effectively Prioritizing Tests in Development Environment. In *Proceedings of the ACM International Symposium on Software Testing and Analysis (ISSTA2002)*, Roma, Italy. pp. 97-106.
 19. K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, R. S. Roos. Time-Aware Test Suite Prioritization. In *Proceedings of the ACM International Symposium on Software Testing and Analysis (ISSTA2006)*, Portland, Maine, USA. pp. 1-12.
 20. Zi-Yuan Wang. "Test Case Generation and Prioritization for Combinatorial Testing". A Dissertation for Ph.D., Southeast University, 2009.
 21. Ziyuan Wang, Baowen Xu, Changhai Nie. "Greedy Heuristic Algorithms to Generate Variable Strength Combinatorial Test Suite". In *Proceedings of the 8th International Conference on Quality Software (QSIC2008)*, Oxford, UK, August 12-13, 2008: 155-160.
 22. Z. Wang, L. Chen, B. Xu, and Y. Huang. "Cost-Cognizant Combinatorial Test Case Prioritization". *International Journal of Software Engineering and Knowledge Engineering*, 2011, 21(06):829-854.
 23. W. E. Wong, J. R. Horgan, S. London, H. Agrawal. A Study of Effective Regression Testing in Practice. In *Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering (ISSRE1997)*: 264-274.
 24. X. Zhang, B. Qu, "An Improved Metric for Test Case Prioritization," In *Proceedings of Web Information Systems and Applications Conference (WISA2011)*: 125-130.