ELEKTROPIKA: Int. J. of Electrical, Electronic Engineering and Technology, Vol. 2, 17–28, 2012

Development of Prioritization Strategy for T-Way Testing

[1]Kamal Z. Zamli*, [2]Rozmie R. Othman, [3]Saidatul K. Said

[1]Faculty of Computer Systems and Software Engineering,
Universiti Malaysia Pahang, Lebuhraya Tun Razak, 26300 Kuantan, Pahang, Malaysia

[2]School of Computer and Communication,
Universiti Malaysia Perlis, PO Box 77, d/a Pejabat Pos Besar, 01007 Kangar, Perlis, Malaysia

[3]School of Electrical and Electronic Engineering,
Universiti Sains Malaysia, Engineering Campus, 14300 Nibong Tebal, Pulau Pinang, Malaysia

*Corresponding Author E-mail: kamalz@ump.edu.my
Tel: +(609)-5492544

Abstract—Software testing is an important phase in the software development life cycle to ensure quality and conformance. Although desirable, exhaustive testing is often problematic owing to time and cost constraints dealing with potentially large test suite size. Thus, researchers develop a sampling technique to resolve the aforementioned problem, termed t-way strategy (where t indicates the interaction strength), based on the systems' parameter interactions. Earlier works suggest that t-way strategy can be helpful to systematically minimize the test suite size without sacrificing its effectiveness for fault detection. As most testing activities happen towards the end of software development, testers are often forced to consider only partial t-way test suite, that is, to be in line with the project deadline. In this situation, there is a need to prioritize the test suite accordingly (i.e. by maximizing interaction coverage in early test cases to ensure high probability for finding errors). Most existing t-way strategies put focus on generating optimal test suite (i.e. covering all desired interactions with the most minimum number of test cases) but less emphasis on the order of coverage. As such, some high coverage interaction faults may be missed to accommodate shift in deadline schedule. In order to alleviate such problems, this paper elaborates on a systematic t-way prioritization strategy, called PriorTC. PriorTC is not only capable to greedily reorganize any existing t-way test suite for prioritization but also can be used to automatically analyze and verify the required interaction coverage.

Keywords: T-Way Testing, Test Case Prioritization, Interaction Testing, Test Suite Verification

1. INTRODUCTION

Software testing is a process of executing any software (using predefined test cases or configurations) with the intent to find faults [1–3]. Consuming over 50% of the development costs [4, 5], software testing is perhaps one of the most important activities within the software development lifecycle [6,7]. Typically, software testing activities involve designing and generating test cases, subjecting the software/module under test with those test cases, and examining the results. To ensure testing effectiveness, one of the key issues is on the design and generation of test cases [8].

The generation of test cases normally involves the application of multiple sampling strategies (including equivalence partitioning, boundary value analysis, cause and effect graphing, and decision tree). Although effective in some application domain, the aforementioned sampling strategies are not sufficiently adaptable for fault due to interactions [8–12]. As a result, many interaction based t-way strategies (e.g. In-Parameter-Order General (IPOG) [13–15], Heuristic IPOG (IPOGF) [16],

Integrated T-way Test Data Generator (ITTDG) [17], Jenny [18] and Test Vector Generator (TVG) [19]) have been proposed in literature. These strategies have been proven effective to generate test suite, however, there are several problems remain.

The first problem relates to the prioritization of the t-way test suite. In search of an optimal strategy (i.e. most interaction elements are covered only once), many of existing t-way strategies have put significant focus on obtaining the smallest test size with minimal execution time but give poor emphasis on test prioritization [22, 23]. As most testing activities happen towards the end of software development, testers are often forced to consider only partial t-way test suite, that is, to be in line with the project deadline. In this situation, there is a need to prioritize the test suite accordingly. Here, prioritization is useful to schedule test case execution in order to maximize interaction coverage, and hence, facilitate fault detection as early as possible [24].

The second problem relates to the completeness of interaction coverage [4, 20, 21]. For small configuration (i.e. with small test suite size), the complete interaction coverage can be verified manually. However, for large configuration (i.e. with potential large test suite size), manual intervention and verification is next to impossible owing to the potentially large number of t-way interaction for consideration [11].

Motivated by these two problems, this paper highlights a useful t-way prioritization strategy, called PriorTC, which can be used to prioritize as well as verify the completeness of interaction coverage. The main novel feature of PriorTC is the fact that it can automatically prioritize any given t-way test suite generated from any existing strategies including that of In-Parameter-Order General (IPOG) [13-15], Heuristic IPOG (IPOGF) [16], Integrated T-way Test Data Generator (ITTDG) [17], Jenny [18] and Test Vector Generator (TVG) [19].

The rest of this paper is organized as follows. Section 2 presents the related work. Section 3 elaborates the PriorTC strategy in details. Section 4 presents assessment results. Finally, Section 5 gives our conclusion.

2. OVERVIEW AND RELATED WORKS

To illustrate the rationale behind t-way testing, consider the option dialog in Microsoft Excel software (see Figure 1). Even if only View tab option is considered, there are already 20 possible configurations to be tested. With the exception of Gridlines color which takes 56 possible values, each configuration can take two values (i.e. checked or unchecked). Here, there are $2^{20} \times 56 = 58,720,256$ combinations of test cases to be evaluated. At the same calculation as in the previous paragraph, it would require nearly 559 years for a complete test for merely View tab option!

The aforementioned example highlights the combinatorial explosion problem. Many alternatives have been explored to address this problem. Parallel testing can be employed to reduce the time required for performing the tests. Nevertheless, as software and hardware are getting more complex than ever, parallel testing approach becomes immensely expensive due to the need for faster and higher capability processors along state-of-the-art computer hardware. Apart from parallel testing, systematic random testing could also be another option. However, systematic random testing tends to dwell on unfair distribution of test cases.

In order to address some of the difficulties in the aforementioned approaches, much research is now focusing on t-way testing strategies. As the name suggests, t-way testing focuses on the t-way interaction of variables (where t indicates the interaction strength). The rationale for interaction testing stemmed from the fact that from empirical observation, the number of input variables involved in software failures is relatively small. If t or fewer variables are known to cause fault, test cases can be generated on some t-way combinations (i.e. resulting into a smaller set of test cases for consideration).

Over the years, many t-way strategies have been proposed in literature strategies including that of IPOG [13-15], IPOGF [16], ITTDG [17], Jenny [18] and TVG [19]). In a nutshell, t-way

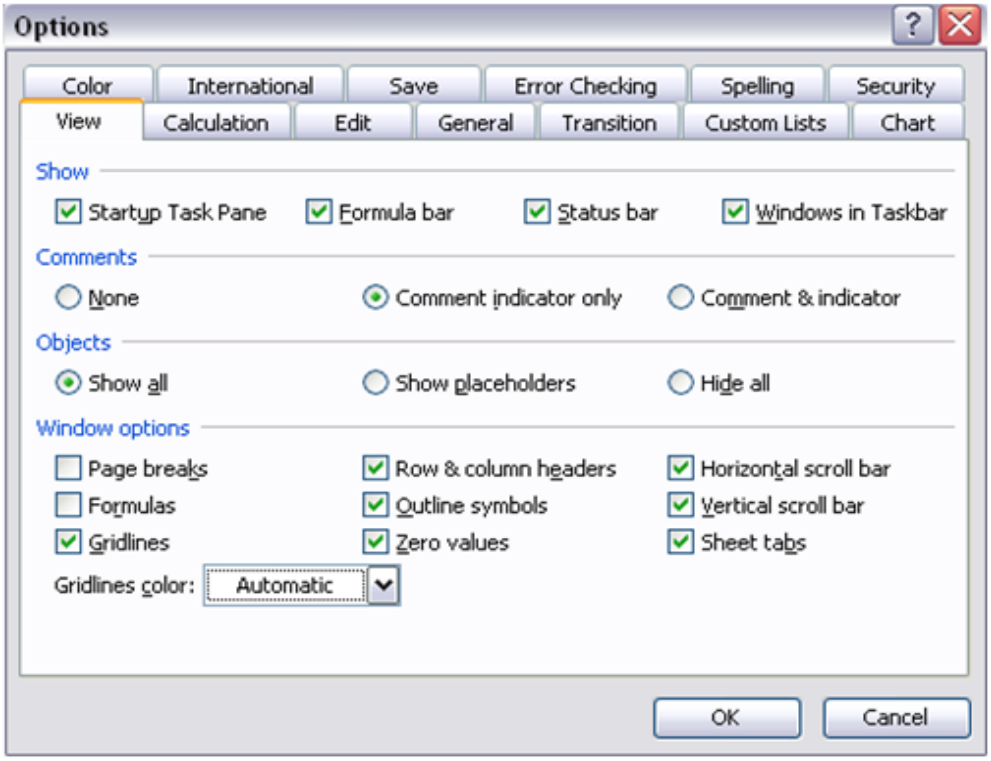

Figure 1: Option Dialog in Microsoft Excel

strategies can be characterized based on the strategy approach for generating the test suite (i.e. either one-parameter-at-a-time (OPAT) strategy or one-test-at-a-time (OTAT) strategy). OPAT strategies usually generate an exhaustive test suite for selected parameters (i.e. the initial test suite) and extend the initial test suite horizontally by adding one uncovered parameter at a time (the process often refers as horizontal extension) until all parameters are covered. In order to ensure the maximum interaction coverage, completely new test cases maybe added during the horizontal extension process and the process is refers as vertical extension. IPOG and IPOGF are examples of OPAT strategies. Both strategies start by generating the exhaustive test suite for the first t parameters and proceed with horizontal as well as vertical extension process. However, both strategies implement horizontal and vertical extension in two different ways. IPOGF horizontal extension only extends test cases (inside the initial test suite) that can produce the most uncovered interaction elements while in IPOG, all test cases in the initial test suite will be extended. As for vertical extension, IPOGF tries to fit the uncovered interaction elements into existing test cases first before creating a completely new test case. Unlike IPOGF, IPOG directly converts the uncovered interaction elements into a new test case.

Contrary to OPAT strategy, OTAT strategy (e.g. ITTDG, Jenny and TVG) generates a complete test case iteratively until all interaction elements are covered by the test suite. In ITTDG, a list of test case candidates is first generated using greedy method. The best test case candidate (i.e. candidate that covers the most uncovered interaction elements) will be selected as the final test cases. Similar to ITTDG, Jenny also implements greedy method in generating test cases. However, unlike ITTDG, Jenny directly generates the final test case without undergoing any candidature process. In addition, Jenny does not directly generate the test suite for t-way testing but instead generates the 1-way test suite first. The 1-way test suite is then extends to cover the 2-way interaction and further extends to cover the 3-way interaction. The extension process is stopped when a t-way interaction

is covered (where t is specified by the user). As for TVG, t-way test suite can be generated using one from the three provided algorithms (i.e. T-Reduced, Plus-One and Random Set). The details on the three algorithms are unclear as there is limited literature regarding the algorithms.

Although most OPAT and OTAT based strategies usefully optimize coverage with minimum number of tests, the generated test cases are often out-of-order as far as interaction coverage is concerned. As such, it is difficult to get high interaction coverage with partial execution of test cases - the practice that is commonly adopted to accommodate tight deadlines. For this reason, research into prioritization is deemed necessary.

3. OVERVIEW OF PRIORTC

PriorTC consists of two main algorithms which are test prioritization algorithm and test verification algorithm (see Figure 2). Both algorithms use the generated test suite (i.e. from any existing t-way strategies) as an input and manipulate it in order to get the desired output. The purpose of test verification algorithm is to analyze the interaction coverage of each test case (i.e. by measuring the coverage ratio (CR) for each test case) while test prioritization algorithm is used to reorganize the test suite so that test case with the highest number of new combination coverage (or CR) is orderly placed in the final test suite.

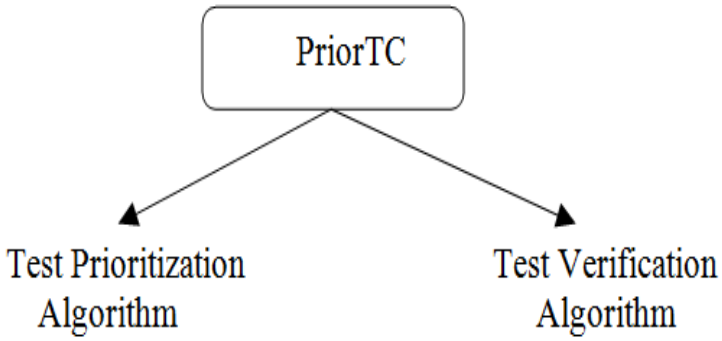

Figure 2: PriorTC

3.1. Test Verification Algorithm

Essentially, any t-way test suite must cover all possible interactions between t input parameters for at least once [11]. However, in search of the optimal test suite, some interacting elements might have accidentally been omitted. In this case, the generated test suite cannot be considered as a complete t-way test suite as some interaction elements are missing. In order to ensure the completeness of any t-way test suite, PriorTC's test verification algorithm can be used to determine the coverage ratio of any given t-way test suite (see equation 1). In doing so, test verification algorithm starts with generated test suite as an input and calculates the number of interaction elements need to be covered in order to form a complete t-way test suite, T. Then, a list of covered interaction elements is created to store the interaction elements that have already been covered. The analysis process starts by decomposing the first test case into interaction elements. The new interaction elements (i.e. interaction elements which have not yet existed in the covered interaction elements list) are pushed into the covered interaction elements list and the number of push interaction elements is recorded as x_1. The coverage ratio for the first test case, CR_1 is calculated by dividing the x_1 with T. The analysis process is repeated for the rest of the test cases and the coverage ratio for each test

case, CR_k, is calculated as follows.

$$CR_k = CR_{k-1} + (\frac{x_k}{T}) \quad (1)$$

For a complete t-way test suite, CR_N should equal to 1 (where N is the number of test cases inside the generated test suite). In general, the test verification algorithm can be summarized in Figure 3.

Input: generated test suite, A
Output: coverage ratio for each test case, CR_k

begin
Calculate T, the total interaction elements need to be covered;
Initialize cEL as covered interaction elements list;
while *A is not empty*
s = pop one test case from A;
decompose s into interaction elements;
push new covered interaction elements into cEL;
calculate CR;
end of while
finish

Figure 3: Test Verification Algorithm

3.2. Test Prioritization Algorithm

Test prioritization algorithm is the core component of PriorTC. Specifically, the aim of test prioritization algorithm is to improve the rate of interaction coverage (or coverage ratio) of any t- way test suite. In order to achieve the aim, test prioritization algorithm reorganizes each test case such that the one with the most number of newly covered interaction elements are located at the top of test suite (i.e. have the higher priority compare to the others). Similar to test verification algorithm, test prioritization algorithm also uses the generated t-way test suite as an input. The first test case is always considered as the first candidate in the prioritized test suite since it covers the highest number of the new interaction elements, *maxWeight*.

After that, the algorithm will search for other test cases that can cover the same number of new interaction elements (or greater) as the *maxWeight*. In doing so, every test case in generated test suite is decomposed into interaction elements and new interaction elements covered by the test case (i.e. not yet covered by the prioritized test suite) are calculated. If the number of new interaction elements is equal or more than *maxWeight*, the test case will be popped from generated test suite on push into prioritized test suite. Towards the end, if there is a test case in generated test suite, the *maxWeight* value will be reduced by 1 and the whole searching process will be repeated. Note that this algorithm does not change the number of original test cases, but reorder the position of the test cases in the final test suite. Overall, the test prioritization algorithm can be summarized in Figure 4.

```
Input: generated test suite, A
Output: prioritized test suite, B

begin
  pop the first test case from A and push into B;
  maxWeight = maximum number of interaction elements can be covered by a test case
  while A is not empty
    for each test cases in A : s = test case
      weight = number of new interaction elements cover by s;
      if weight >= maxWeight
        pop s from A and append into B;
      end if
    end for each
    reduce maxWeight by 1;
  end while
finish
```

Figure 4: Test Prioritization Algorithm

4. EVALUATIONS

The main goals of our evaluation are twofold. Firstly, we wish to demonstrate the applicability of PriorTC for ensuring complete interaction coverage. Secondly, we wish to demonstrate the capability of PriorTC for improving the rate of interaction coverage of any t-way test suite (i.e. prioritizing the t-way test suite). To achieve the goals, two experiments have been designed and carried out. Both experiments will be discussed in the two upcoming sub-sections respectively. Here 6 existing t-way strategies (i.e. IPOG [13], IPOF [16], ITTDG [17], Jenny [18], PICT [25] and TVG [19]) are used as t-way test suite generator for both experiments. The rationale for adopting these strategies stemmed from the fact that their implementations are readily available as public domain tools.

To undertake the experiments, we have adopted the hardware configurations consisting of a desktop PC with Windows XP, 2.8 GHz Core 2 Duo CPU, 1 GB of RAM. The PriorTC strategy is designed and implemented in Java (JDK 1.6).

4.1. Applicability of PriorTC for Interaction Coverage Analysis

For the first experiment, the aforementioned t-way strategies (i.e. IPOG, IPOF, ITTDG, Jenny, PICT and TVG) are summoned in our running environment to generate a 2-way test suite for a 20 10-valued parameters system. A total of 19,000 interaction elements need to be covered by every test case in order to form a complete 2-way test suite. Using PriorTC's test verification algorithm, the coverage ratio for each test case generated by every strategy is recorded.

Table 1 summarizes the output from PriorTC's test verification algorithm. It should be noted that in total IPOG generated 220 test cases, IPOGF generated 203 test cases, ITTDG generated 195 test cases, Jenny generated 194 test cases, PICT generated 216 test cases and TVG generated 247 test cases. Comparatively, Jenny produces the most minimal test size.

Referring to Table 1, most strategies cover all interaction elements required for a 2-way interaction (i.e. by achieving coverage ratio 1) with the exception of Jenny. Thus, it can be

Table 1: 2-way Test Suite for a System with 20 10-valued parameters

CR	Test Case Number					
	IPOG	IPOGF	ITTDG	Jenny	PICT	TVG
0.1	10	10	10	10	10	10
0.2	21	21	21	21	21	21
0.3	32	32	32	32	32	32
0.4	44	44	43	44	44	44
0.5	57	,57	56	57	57	58
0.6	71	70	69	71	72	73
0.7	88	88	84	87	89	92
0.8	107	105	103	107	109	115
0.9	138	137	130	134	139	149
1.0	220	203	195	Not reach	216	247

concluded that Jenny's test suite is incomplete as some interaction elements are missing from the test suite.

Another subtle observation is the fact that PriorTC's test verification algorithm can also provide testers with insights into the coverage information when only parts of the test suite are executed (i.e. demonstrating the gradual greediness of each of the strategies).Comparatively, when the first 84 test cases generated by ITTDG are executed, 70% of total 2-way interaction elements are tested. To achieve the same number of interaction coverage of 70% would require 88 test cases for IPOG and IPOF, 87 test cases for Jenny, 89 test cases for PICT, and 92 test cases for TVG respectively. For this particular scenario, ITTDG gives the best performance as far as the number of test cases is concerned.

4.2. Capability of PriorTC for Test Prioritization

For the second experiment, we would like to demonstrate the capability of PriorTC for prioritizing t-way test suite. Ideally, prioritized test suite gives better coverage ratio as compared to the original test suite. In this experiment, a 4-way test suite is generated for a TCAS system. TCAS (stands for Traffic Collisions and Avoidance System) is a system used to warn pilot for any potential collisions between aircrafts. This system consists of 2 10-valued parameters, 1 4-valued parameters, 2 3-valued parameters and 7 2-valued parameters. Using PriorTC test verification algorithm, coverage ratio for each test case generates by every strategies for originally generated test suite is recorded. Table 2 summarizes the result.

Then every generated test suite is prioritized by PriorTC's test prioritization algorithm. Again, the coverage ratio for prioritized test suite is analyzed using PriorTC's test verification algorithm. The summary of analysis is depicted in Table 3.

Based on the results from Table 2 and 3, we can make a comparison of interaction coverage rate between generated and prioritized data. To facilitate comparison, coverage ratio graph between generated and prioritized test suite for IPOG, IPOGF, ITTDG, Jenny, PICT and TVG are shown in Figure 5, 6, 7, 8, 9 and 10 respectively.

From the plotted graphs, the obvious improvement can be seen in the case of IPOG and IPOGF where the prioritized test suite covers faster than the original test suite. For IPOG, testers only need to run 1370 test cases in order to achieve maximum interaction coverage with prioritized test suite whereas for the original test suite, testers need to execute 1377 test cases in order to achieve

Table 2: Coverage Ratio for Generated Test suite

CR	Test Case Number					
	IPOG	IPOGF	ITTDG	Jenny	PICT	TVG
0.1	17	18	17	14	14	14
0.2	50	51	49	31	31	30
0.3	100	100	98	53	53	51
0.4	165	171	160	83	82	78
0.5	273	289	268	130	120	113
0.6	457	465	453	195	173	163
0.7	617	620	616	314	251	236
0.8	832	830	829	529	374	356
0.9	987	987	985	949	610	562
1.0	1377	1366	1309	1548	1454	1589

Table 3: Coverage Ratio for Prioritized Test Suite

CR	Test Case Number					
	IPOG	IPOGF	ITTDG	Jenny	PICT	TVG
0.1	14	14	14	15	14	14
0.2	30	30	31	33	30	30
0.3	52	51	52	53	52	51
0.4	79	78	80	80	78	78
0.5	114	114	115	120	114	113
0.6	164	163	166	174	163	163
0.7	238	238	246	256	237	236
0.8	363	358	383	386	359	356
0.9	594	581	620	645	583	574
1.0	1370	1362	1309	1548	1450	1571

the same coverage. Concerning IPOGF, testers need to run 1362 test cases to achieve maximum interaction coverage using prioritized test suite. However, testers need to run 1366 test cases using original test suite is used. For other strategies, the increase in interaction coverage occurs in a small scale (especially for PICT and TVG). This is mainly due to the level of greediness of their implemented algorithms.

5. CONCLUSION

In order to achieve the faster combination coverage, the importance of test prioritization is highlighted in this paper. This paper also highlights the potential cost saving as far as test execution time is concerned when the test suite is prioritized accordingly. As the scope for future work, we are planning to complete our first prototype with GUI support and make the implementation available in the public domain so that it can be shared by the software engineering community at large.

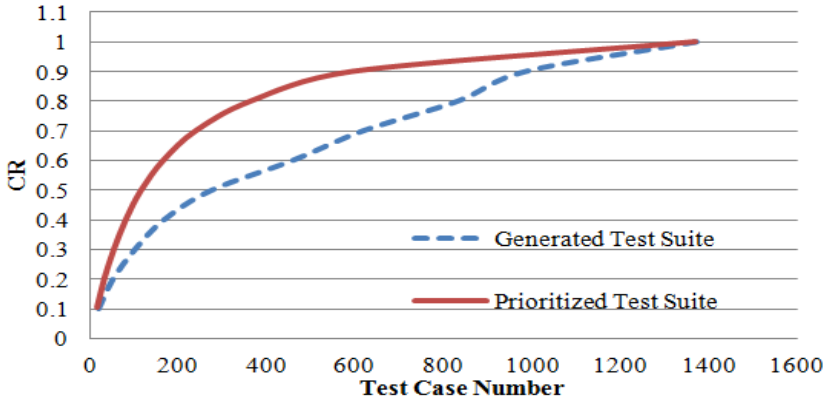

Figure 5: Graph of comparison generated and prioritized test suite for IPOG

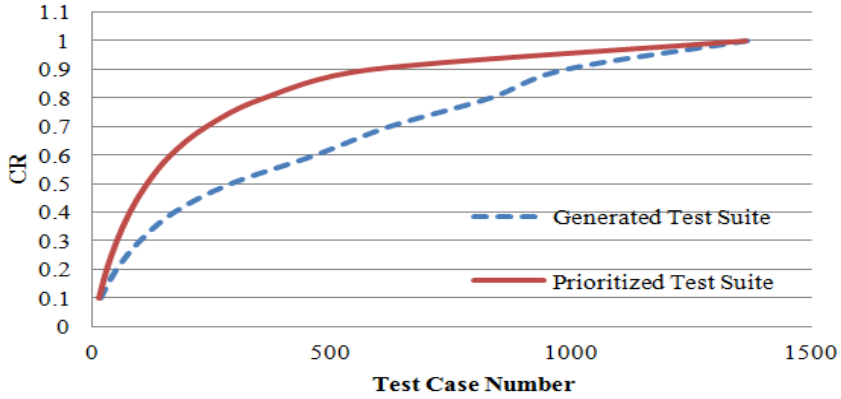

Figure 6: Graph of comparison generated and prioritized test suite for IPOGF

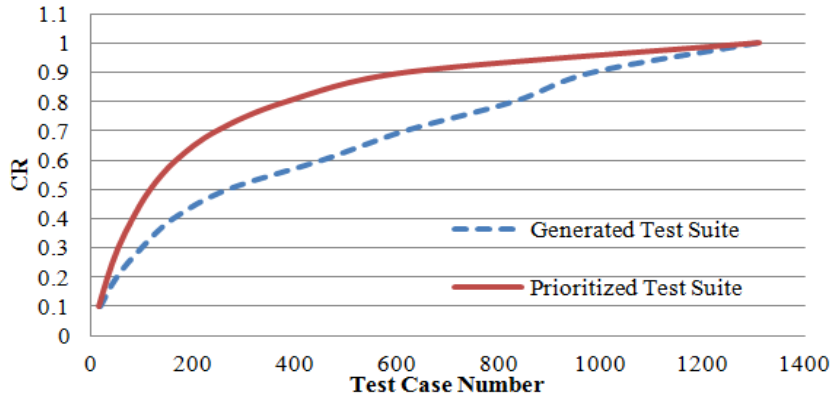

Figure 7: Graph of comparison generated and prioritized test suite for ITTDG

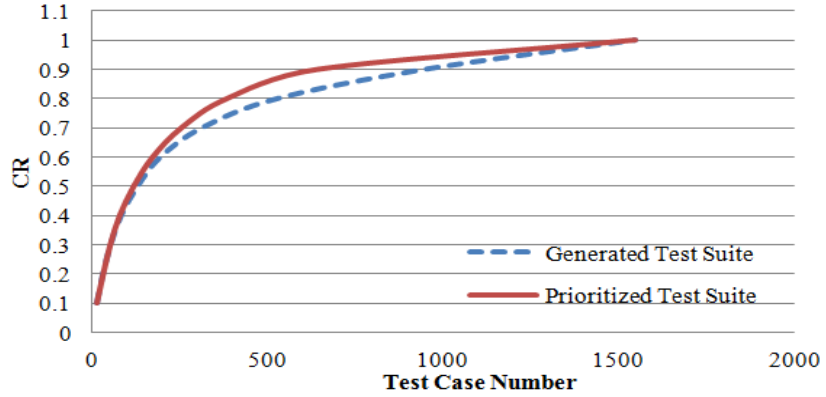

Figure 8: Graph of comparison generated and prioritized test suite for Jenny

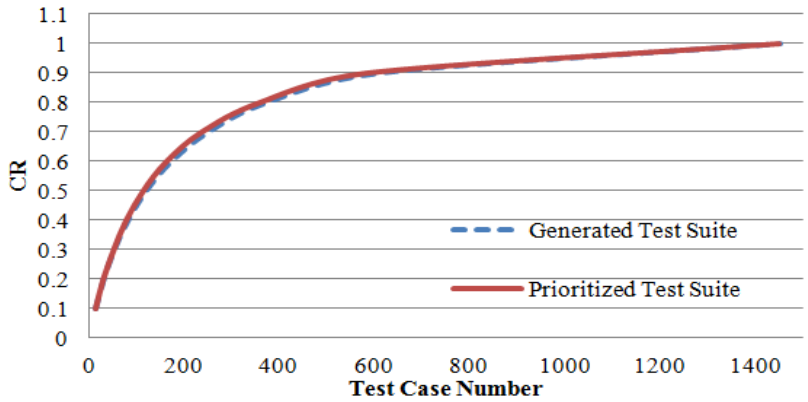

Figure 9: Graph of comparison generated and prioritized test suite for PICT

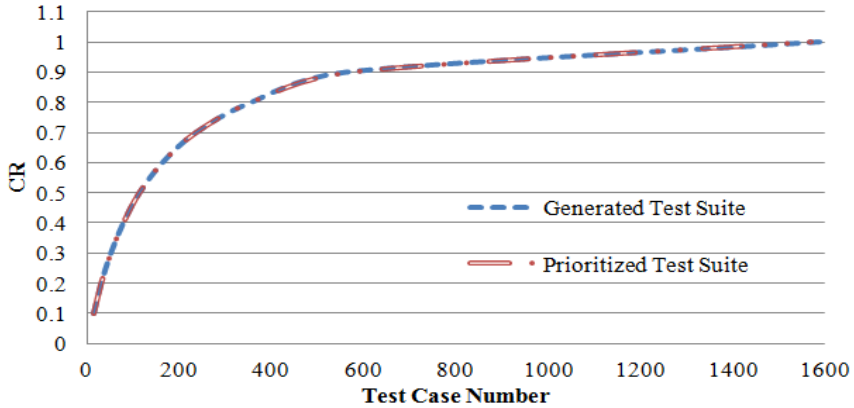

Figure 10: Graph of comparison generated and prioritized test suite for TVG

ACKNOWLEDGMENT

This research is partial funded by the generous Universiti Malaysia Pahang, Short Term Grants - "Development of a Pairwise Interaction Testing Strategy with Check-Pointing Recovery Support".

REFERENCES

1. Myers, G.J. 2004. *The Art of Software Testing*, New Jersey: John Wiley & Sons Inc.
2. Ahmed, B.S. 2012. Adopting a Particle Swarm-Based Test Generator Strategy for Variable-Strength and T-way Testing, *PhD Thesis*, School of Electrical And Electronic Eng., Universiti Sains Malaysia.
3. Ahmed, B.S. and Zamli, K.Z. 2011. A Variable Strength Interaction Test Suites Generation Strategy Using Particle Swarm Optimization. *Journal of Systems and Software*, 84(12):2171-2185.
4. Zamli, K.Z. et al. 2008. *Software Testing*. Kuala Lumpur: Open University Malaysia.
5. Grindal, M. et al. 2005. Combination Testing Strategies: A Survey. *Journal of Software Testing, Verification and Reliability*, 15(3):167-199.
6. Carroll, C.T. 2003. The Cost of Poor Testing: A U.S. Government Study (Part 1). *EDPACS: The EDP Audit, Control, and Security Newsletter*, 31(1):1-17.
7. Carroll, C.T. 2003. The Cost of Poor Testing: A U.S. Government Study (Part 2). *EDPACS: The EDP Audit, Control, and Security Newsletter*, 31(2):1-16.
8. Nie, C. and Leung, H. 2011. A Survey of Combinatorial Testing. *ACM Computing Surveys*, 43(2).
9. Klaib, M.F.J. 2009. Development Of An Automated Test Data Generation And Execution Strategy Using Combinatorial Approach. *PhD Thesis*, School of Electrical And Electronic Eng., Universiti Sains Malaysia.
10. Zamli, K.Z. et al. 2011. Design And Implementation Of A T-Way Test Data Generation Strategy With Automated Execution Tool Support. *Information Sciences*, 181(9):1741-1758.
11. Younis, M.I. 2010. MIPOG: A Parallel T-Way Minimization Strategy For Combinatorial Testing. *PhD Thesis*, School of Electrical And Electronic Eng., Universiti Sains Malaysia.
12. Younis, M.I. et al. 2008. MIPOG - Modification Of The IPOG Strategy For T-Way Software Testing. *Proceeding of The Distributed Frameworks and Applications (DFmA)*, Penang, Malaysia.
13. Lei, Y. et al. 2007. IPOG: A General Strategy For T-Way Software Testing. *Proceedings of the 14th Annual IEEE International Conference and Workshops on The Engineering of Computer-Based Systems*, Tucson, AZ, pp. 549-556.
14. Lei, Y. et al. 2008. "IPOG/IPOG-D: Efficient Test Generation For Multi-Way Combinatorial Testing. *Journal of Software Testing, Verification and Reliability*, 18(3):125-148.
15. Lei, Y. and Tai, K.C. 1998. In-Parameter-Order: A Test Generation Strategy For Pairwise Testing. *Proceedings of 3rd IEEE International Conference on High Assurance Systems Engineering Symposium*, Washington DC, USA, pp. 254-261.
16. Forbes, M. et al. 2008. Refining The In-Paramater-Order Strategy For Constructing Covering Arrays. *Journal of Research of the National Institute of Standards and Technology*, 113(5):287-297.
17. Othman, R.R. and Zamli, K.Z. 2011. ITTDG: Integrated T-way Test Data Generation Strategy for Interaction Testing. *Scientific Research and Essays*, 6(17):3638-3648.
18. Jenkins, B. Jenny Test Tool. Available: `http://www.burtleburtle.net/bob/math/jenny.html` (Last Accessed:- 2010, February)

19. Arshem, J. TVG. Available: `http://sourceforge.net/projects/tvg` (Last Accessed:- 2010, December)
20. Othman, R.R. and Zamli, K.Z. 2011. T-Way Strategies and Its Applications for Combinatorial Testing. *International Journal of New Computer Architectures and their Applications*, 1(2):459-473.
21. Zamli, K.Z. et al. 2011. Practical Adoptions of T-Way Strategies for Interaction Testing. *Software Engineering and Computer Systems*, vol 181, J. M. Zain, et al., Eds., ed: Springer Berlin Heidelberg, pp. 1-14.
22. Bryce, R.C. and Colbourn, C.J. 2006. Prioritized Interaction Testing for Pairwise Coverage with Seeding and Avoids. *Journal of Information and Software Technology*, 48(10):960-970.
23. Cohen, M.B. et al. 2003. Constructing Test Suites for Interaction Testing. *Proceedings of 25th IEEE International Conference on Software Engineering*, Portland, Oregon, USA, pp. 38-48.
24. Qu, X. et al. 2007. Combinatorial Interaction Regression Testing: A study of Test Case Generation and Prioritization. *Proceedings of the IEEE International Conferance on Software Maintenance (ICSM)*, Paris, France, pp. 413-418.
25. Czerwonka, J. PICT Tool. Available: `http://www.pairwise.org/tools.asp` (Last Accessed:- 2010, February)