Pairwise Test Case Generation for Less Number of Test- Case Sets

Shalini Gupta
Virendra Swarup Group of Institutions, Kanpur, India

Avdhesh Gupta
IMS Engineering College, Ghazaibad

ABSTRACT
Software testing is the process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software items. Software testing is an activity that should be done throughout the whole development process. Pairwise testing primarily targets faults caused by interactions between two parameters. However, some faults can be caused by interactions involving more than two parameters. Those faults cannot effectively be detected by pairwise testing. In this research work, we presented an algorithm to generate effective and less number of test cases using pairwise testing technique. The pairwise testing approach is basically based on the fact that the majority of possible errors/faults/bugs occur when two modules/parameters values interact. This proposed algorithm can be used efficiently in various realms of software products. In future we can plan to reduce the number of test cases by using the degree of coverage of three and four-wise in efficient way. Ultimately this will reduce the total number of test cases and provide only effective and efficient test case set and thus it will also save time for both software developers as well as for software testers.

Keywords
Pairwise testing, software testing, graph base testing.

1. INTRODUCTION
Why test software? To find the bugs! is the instinctive response and many people, developers and programmers included, think that that‘s what debugging during development and code reviews is for, so formal testing is redundant at best. But a bug is really a problem in the code; software testing is focused on finding defects in the final product. Here are some important defects that better testing would have found.

First, test what‘s important. Focus on the core functionality, the parts that are critical or popular before looking at the _nice to have features. Concentrate on the application‘s capabilities in common usage situations before going on to unlikely situations. For example, if the application retrieves data and performance are important, test reasonable queries with a normal load on the server before going on to unlikely ones at peak usage times. It‘s worth saying again: focus on what‘s important. Good business requirements will tell you what‘s important.

The value of software testing is that it goes far beyond testing the underlying code. It also examines the functional behavior of the application. Behavior is a function of the code, but it doesn‘t always follow that if the behavior is ―bad‖ then the code is bad. It‘s entirely possible that the code is solid but the requirements were inaccurately or incompletely collected and communicated. It’s entirely possible that the application can be doing exactly what we‘re telling it to do but we‘re not telling it to do the right thing.

Software testing is not a one person job. It takes a team, but the team may be larger or smaller depending on the size and complexity of the application being tested. The programmer(s) who wrote the application should have a reduced role in the testing if possible. The concern here is that they‘re already so intimately involved with the product and know that it works that they may not be able to take an unbiased look at the results of their labors. Pairwise testing primarily targets faults caused by interactions between two parameters. However, some faults can be caused by interactions involving more than two parameters. Those faults cannot effectively be detected by pairwise testing. Pairwise testing is black box testing technique. It is a strategy in which testing is based solely on the requirements and specifications. This testing requires no knowledge of the internal paths, structure, or implementation of the software under test (SUT). This significantly reduces the number of tests that must be created and run and these test cases are also manageable so that any novice software tester can easily operate it against software applications.

2. LITERATURE REVIEW

Ljubomir Lazic, et al said that organizations are constantly working to leverage today‘s best practices for testing within the context of their existing IT environments. As IT works to balance the business needs for a certain application and the testing limitations with regards to resources and schedules, making the best use of the testing environment becomes critical. Optimized testing is a way for organizations to move their testing efforts forward to reflect changing business environments and resource constraints. Optimized testing uses test techniques which has the highest defect detection yield and combined with the Orthogonal Array Testing Strategy (OATS) provides:

1. Pairwise testing that protects against pairwise bugs while dramatically reducing the number of tests to perform which is especially cool because pairwise bugs represent

the majority of combinatory bugs and such bugs are a lot more likely to happen than the ones that only happen with more variables.

2. Plus, the availability of tools means you no longer need to create these tests by hand.
3. Pairwise testing might find some pairwise bugs while dramatically reducing the number of tests to perform, compared to testing all 34 combinations because pairwise bugs represent the majority of combinatory bugs.
4. Plus, the availability of tools means you no longer need to create these tests by hand, except for the work of analyzing the product, selecting variables and values, actually configuring and performing the test, and analyzing the results which improves application quality, maximizes development resources and helps deliver applications on time and within budget.

The author had found a method that he enjoyed so much that he used it and talk about it as often as possible. He had seen this technique referred to as Pairwise Testing.

James Bach et al said that pairwise testing is a wildly popular approach to combinatorial testing problems. The number of articles and textbooks covering the topic continue to grow, as do the number of commercial and academic courses that teach the technique. Despite the technique's popularity and its reputation as a best practice, the author found the technique to be over promoted and poorly understood. In this paper, he defined pairwise testing and review many of the studies conducted using pairwise testing. Based on these studies and our experience with pairwise 39 testing, he discussed weaknesses he perceived in pairwise testing. Knowledge of the weaknesses of the pairwise testing technique, or of any testing technique, is essential if he was to apply the technique wisely. He concluded by re-stating the story of pairwise testing and by warning testers against blindly accepting best practices.

Pairwise testing protects against pairwise bugs while dramatically reducing the number tests to perform, which is especially cool because pairwise bugs represent the majority of combinatoric bugs, and such bugs are a lot more likely to happen than ones that only happen with more variables. Plus, you no longer need to create these tests by hand.

Pairwise testing might find some pairwise bugs while dramatically reducing the number tests to perform, compared to testing all combinations, but not necessarily compared to testing just the combinations that matter which is especially cool because pairwise bugs might represent the majority of combinatoric bugs, or might not, depending on the actual dependencies among variables in the product and some such bugs are more likely to happen than ones that only happen with more variables, or less likely to happen, because user inputs are not randomly distributed. Plus, you no longer need to create these tests by hand, except for the work of analyzing the product, selecting variables and values, actually configuring and performing the test, and analyzing the results.

3. PAIRWISE Testing

Consider that a software object has n input parameters, each parameter having d possible values. One straightforward approach to testing this object is to test every possible n-way combination of values; for instance, every combination of values of the n parameters. This approach covers the entire input space, but is nearly always impractical for real-world software due to the well-known combinatorial explosion problem. The idea of pairwise testing is already 20 years old but for the last five years its popularity has been rising extremely. The reason is that testers have to face more complex software projects with the same time target. Pairwise testing is an alternative approach that only tries to test every possible two-way combination of values; that is, every combination of values of any two parameters. Testing all two-way combinations, instead of all n-way combinations, does not cover the entire input space. However, empirical studies show that many software faults are caused by interactions between only two parameters. Testing all two-way combinations can effectively detect these faults, while substantially reducing the number of tests.

There is much unreliable evidence about the benefit of pairwise testing. Unfortunately, there are only a few documented studies:

1. In a case study published by Brownlie of AT&T regarding the testing of a local-area network-based electronic mail system, pairwise testing detected 28 percent more defects than their original plan of developing and executing 1,500 test cases (later reduced to 1,000 because of time constraints) and took 50 percent less effort.
2. A study by the National Institute of Standards and Technology published by Wallace and Kuhn on software defects in recalled medical devices reviewed fifteen years of defect data. They concluded that 98 percent of the reported software flaws could have been detected by testing all pairs of parameter settings.
3. Kuhn and Reilly analyzed defects recorded in the Mozilla Web browser database. They determined that pairwise testing would have detected 76 percent of the reported errors

3.1 Proposed Algorithm

The proposed algorithm is an effective solution for test case generation. It is designed in order to obtain less and effective no of test case set. Whenever a software product is launched into the market, there is no guarantee whether this product will provide the same result as per client expectations until tested against effective test cases. If the software developer really wants to launch error free software product, then he has to test it against good test cases.

In this thesis work, we have proposed an efficient algorithm which is based on pairwise testing technique. As we know that the behaviour of a software application may be affected by many parameters, e.g., input parameters, environment configurations, and state variable. It is impractical to test all possible combinations of values of all those parameters. So instead of testing all possible combinations, I considered only a subset of well defined combinations of parameter values to test the module in effective way.

It is also observed that every parameter don't participate in every defects, and it is often the case that a defect is caused by interactions among few parameters.

By the help of this technique the tester will generate effective set of test cases .If the software module is tested against these test cases, then the software developer need not worry about its performance when it is used by the client side. Also this is very easy to use these test cases against the software module. With the help of this Algorithm I tried to remove unused test cases and reduced testing time. In short, pairwise testing is a technique that allows user to reduce a large, unmanageable set of test-case inputs to a much smaller set that is likely to reveal bugs in the system under test.

Step1: Begin.
Step2: Read input file that contains parameters and their values.
Step3: Create an empty test case set.
Step4: Construct all pairs of parameter values**.**
Step5: Select only unique pairs.
Step6: Combine these pairs to form test case.
Step 7: Add these test case to test case set.
Step8: Display test case set.
Step9: End.

Our proposed algorithm is based on greedy approach. This is simple and straight forward. In this approach we can take decisions on the basis of information in hand without worrying about the effect these decisions may have in future. That's why the proposed algorithm is easy to implement and most of the time quite efficient. This algorithm starts with a locally optimal choice, and continues making locally optimal choices until complete set of test cases is found.

According to algo which stands for Modified Pair Test Case Generation algorithm, first we read an input file which contains different number of parameters and their different values. After that we make an empty set E, which will contain resultant and efficient test cases later. Now construct all possible 67 pairs of given parameters values. We select only different pairs of previous constructed pairs. Now combine these pairs to form test case and add it to empty set E, until all different pairs are covered by at least one test case. Ultimately we will get resultant and efficient test case set generated by algorithm.

Consider three parameters P1, P2 and P3 and their values. Each parameter has two values. Parameter P1 has value a and b while parameter P2 contains c, d and P3 has e and f respectively.

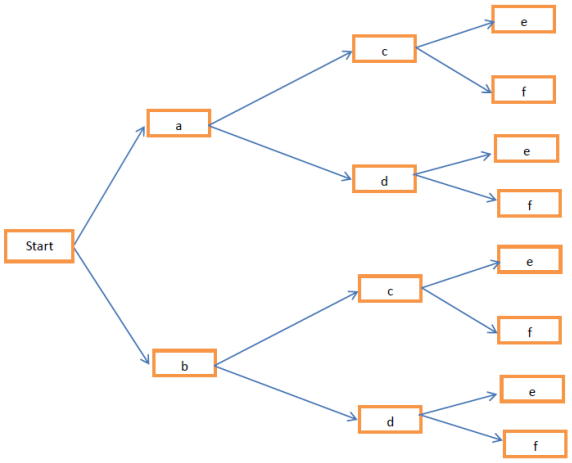

Figure 1: Tree Diagram for Test Case

Here total no of parameter values =6
No of Unique pairs are: 12
These are as follows:
{ (a,c),(a,e),(a,f),(a,d) 69 (b,c),(b,e),(b,d),(b,f)
(c,e),(c,f),(d,e),(d,f) }
Now resultant test set which contains all unique set is:
{ (a,c,e) (a,d,f) (b,c,e) (b,d,f) }

Using MPTCG algorithm, total number of test cases are reduced to 4 which was originally 8. It shows that proposed algorithm works well and helpful to generate test case which are essential for verification and validation of software product. It will also increase the reliability of the software component.

4. Result Comparison

After testing our proposed algorithm, I found positive result comparable to other existing algorithms based on pairwise testing. Here the output comparison of test suite generated by basic pairwise (PW), AETG, and our proposed algorithm. The comparison table is shown below.

Table 1: Comparison of test suite size (basic pairwise, AETG and proposed also for equally-sized sets.

Sets	No. of Elements	Test Suite		
		PW	AETG	New Algo
3	5	28	28	29
3	6	40	41	39
3	7	53	53	57
3	8	64	64	70
3	9	88	88	89
3	10	112	115	112
4	11	142	148	144
4	12	144	176	173
4	13	194	209	204
4	14	224	234	232
4	15	254	265	272
5	10	130	137	127

5. CONCLUSION

In this research work, we presented an algorithm to generate effective and less number of test cases using pairwise testing technique. The pairwise testing approach is basically based on the fact that the majority of possible errors/faults/bugs occur when two modules/parameters values interact. This proposed algorithm can be used efficiently in various realms of software products. According to our own knowledge, the proposed algorithm is quite efficient which covers almost all different parameter values. But as the no of input parameters and their values increases, there might be some problem. These problems are how to handle such large test cases which increase as the parameter and their values increase.

6. FUTURE WORK

So in our future work we plan to reduce the number of test cases by using the degree of coverage of three and four-wise in efficient way. Ultimately this will reduce the total number of test cases and provide only effective and efficient test case set and thus it will also save time for both software developers as well as for software testers.

6. REFERENCES

[1] Pedro Flores, Yoonsik Cheon, PWiseGen: Generating Test Cases for Pairwise Testing Using Genetic Algorithms, IEEE International Conference on Computer Science and Automation Engineering (CSAE 2011), Shanghai, China, June 10-12, 2011.

[2] Jacek Czerwonka, Pairwise Testing in Real World Practical Extensions to Test Case Generators, Microsoft Corporation, February 2008.

[3] Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, Yves le Traon Lassy, Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines, Proceeding ICST '10 Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation, Pages 459-468, EEE Computer Society Washington, DC, USA,2010.

[4] James Bach, Patrick J. Schroeder, Pairwise Testing: A Best Practice That Isn‘t, 22nd Annual Pacific Northwest Software Quality Conference, 2004.

[5] Kuo-Chung Tai, Yu Lei, A Test Generation Strategy for Pairwise Testing, IEEE Transaction on Software Engineering, Volume 8, No. 1, Washington, DC, 13 November 1998.

[6] Kevin Burr, William Young, Combinatorial Test Techniques: Table-based Automation, Test Generation and Code Coverage, Software Engineering Analysis Lab, Nortel. 82

[7] Jerry Huller, Reducing Time to Market with Combinatorial Design Method Testing, USA, December 2005.

[8] G. Bernet, L. Bouaziz, and P. LeGall, A Theory of Probabilistic Functional Testing, Proceedings of the 1997 International Conference on Software Engineering, pp. 216 –226,1997.

[9] B. Beizer, Software Testing Techniques, Second Edition, Van Nostrand Reinhold Company Limited, 1990.

[10] S. Beydeda and V. Gruhn, An integrated testing technique for component-based software, ACS/IEEE International Conference on Computer Systems and Applications, pp 328 – 334, June 2001.

[11] A. Bertolino, P. Inverardi, H. Muccini, and A. Rosetti, An approach to integration testing based on architectural descriptions, Proceedings of the IEEE ICECCS- 97, pp. 77-84, 1997.

[12] J.B. Good Enough and S. L. Gerhart, Toward a Theory of Test Data Selection, IEEE Transactions on Software Engineering, pp. 156-173, June 1997.

[13] D. Gelperin and B. Hetzel, The Growth of Software Testing, Communications of the ACM, Volume 31 Issue 6, pp. 687-695, June 1988.

[14] J. Hartmann, C. Imoberdorf, and M.Meisinger, UML-Based Integration Testing, Proceedings of the International Symposium on Software Testing and Analysis, ACM SIGSOFT Software Engineering Notes, August 2000.

[15] W. E. Howden, Functional Testing and Design Abstractions, The Journal of System and Software, Volum 1, pp. 307-313, 1980. 83

[16] P. Jalote and Y. R. Muralidhara, A coverage based model for software reliability estimation, Proceedings of First International Conference on Software Testing, Reliability and Quality Assurance, pp. 6 –10 (IEEE), 1994.

[17] E. F. Miller, ―Introduction to Software Testing Technology, Tutorial: Software Testing & Validation Techniques, Second Edition, IEEE Catalog No. EHO 180-0, pp. 4-16.

[18] D. Richardson, O‘Malley and C. Tittle, Approaches to specification-based testing, ACM SIGSOFT Software Engineering Notes, Volume 14 , Issue 9, pp. 86 – 96 1989.

[19] S. Redwine & W. Riddle, Software technology maturation, Proceedings of the Eighth International Conference on Software Engineering, pp. 189-200, May 1985.

[20] M. Shaw, Prospects for an engineering discipline of software, IEEE Software, pp. 15-24, November 1990.

[21] L. J. White and E. I. Cohen, A Domain Strategy for Computer Program Testing, IEEE Transactions on Software Engineering, pp. 247-257, May 1980.

[22] J. A. Whittaker, What is Software Testing? And Why Is It So Hard? IEEE Software, pp. 70-79, January 2000.

[23] R. Mandl. Orthogonal Latin Squares: An application of experiment design to compiler testing. Communications of the ACM, 28(10):1054-1058, October 1985. 84

[24] Y. Lei and K.C. Tai. In-parameter-order: A test generation strategy for pair-wise testing. In Proceedings of the third IEEE High Assurance Systems Engineering Symposium, pages 254-261. IEEE, November 1998.

[25] D.M. Cohen, S.R. Dalal, J. Parelius, and G.C. Patton. The Combinatorial Design Approach to Automatic Test Generation. IEEE Software, pages 83-88, September 1996.

[26] A.W. Williams and R.L. Probert. A practical strategy for testing pair-wise coverage of network interfaces. In Proceedings of the 7th International Symposium on Software Reliability Engineering (ISSRE96), White

Plains, New York, USA, Oct 30 - Nov 2, 1996, Nov 1996.

[27] A.W. Williams. Determination of test configurations for pair-wise interaction coverage. In Proceedings of the 13th International Conference on the Testing of Communicating Systems (TestCom2000), Ottawa, Canada, August 2000, pages 59-74, August 2000.

[28] Karen Meagher. Covering Arrays on Graphs: Qualitative independence Graphs and extremal Set partition Theory. Chapter 2.

[29] Mats Grindal, Jeff Offutt, Sten F. Andler. Combination Testing Strategies: A Survey. GMU Technical Report ISE-TR-04-05, July 2004

[30] N.P. Kropp, P.J. Koopman, and D.P. Siewiorek. Automated robustness testing of off-the-shelf software components. In Proceedings of FTCS'98: Fault Tolerant Computing Symposium, June 23-25, 1998 in Munich, Germany, pages 230-239. IEEE, 1998.

[31] Macario Polo Usaola and Beatriz Pérez Lamancha, A framework and a web implementation for combinatorial testing.