An Enhanced Pairwise Search Approach for Generating Optimum Number of Test Data and Reduce Execution Time

Dr. Mohammod Abul Kashem
Faculty of Computer Science and Engineering
Dhaka University of Engineering and Technology (DUET)
Gazipur, Dhaka, Bangladesh
drkashem11@duet.ac.bd

Mohammad Naderuzzaman (Corresponding author)
Department of Computer Science and Engineering
Dhaka University of Engineering and Technology (DUET)
Gazipur, Dhaka, Bangladesh
nader_u@yahoo.com

Abstract: In recent days testing considers the most important task for building software that is free from error. Since the resources and time is limited to produce software, hence, it is not possible of performing exhaustive tests (i.e. to test all possible combinations of input data.) An alternative to get ride from this type exhaustive numbers and as well to reduce cost, an approach called Pairwise (2 way) test data generation approach will be effective. Most of the software faults in pairwise approach caused by unusual combination of input data. Hence, the demand for the optimization of number of generated test-cases and reducing the execution time is growing in software industries. This paper proposes an enhancement in pairwise search approach which generates optimum number of input values for testing purposes. In this approach it searches the most coverable pairs by pairing parameters and adopts one-test-at-a-time strategy for constructing a final test-suite. Compared to other existing strategies, Our proposed approach is effective in terms of number of generated test cases and of execution time.

Keywords:, Software testing, Pairwise testing, Combinatorial interaction testing, Test case generation.

1 Introduction

In software engineering, software testing and debugging is the integral part of software development life cycle, but this process is very much labor-intensive as well expensive [1]. Currently in any software development project around 50% money goes for the software testing. Because of this problem, the focus is to find ways for software testing which will be automatic and cost-effective and to find debugging techniques to ensure high quality of released software product [2]. Current researches on software testing focuses on test-case generation problem, test coverage criterion design, regression testing problem, test oracle problem and fault localization problem [1]. Among all these problems, test-case generation problem is one of most important issue to reduce cost and effort to produce error free software [1]. Pairwise strategy (i.e. two-way interaction) has been known as an effective test case reduction strategy is used to solve this problem. This strategy is able to detect 60 to 80 percent of the faults [4].

As an example, let us take the 'proofing' tab under 'option' dialog box in Microsoft word 2007, there are maximum 13 possible configuration needed to be tested. And each configuration has two values (checked or unchecked) to choose from, other than these, the 'French Modes' have 3 possible values and also there are 'Writing Style' mode has 2 possible values. In this case to test this proofing tab exhaustively, the number of test cases needs to be executed are 2^{13} x 2 x 3 i.e. 49,152. Let us assume that in each test case may consume 4 minutes to execute in average; which results around 136 days to complete the exhaustive test for this tab [3,4].

The case is similar as the hardware products test. Let us say, if a product has 20 on/off switches, for testing all possible combination it may need 2^{20} = 104,85,76 test cases, and may take around 8 year to complete the test by considering 4 minutes for each single test case [4]. Current trend of research work in combinatorial testing aims for generating the least possible test cases [5]. Non-deterministic polynomial-time hard (NP-hard) may be the solution of this problem [6]. Till today many approaches have been proposed by many scientists and also many tools have been developed for finding out the least possible test suits in polynomial time [5-8, 10-14] but yet to find the most optimum one. In this paper we introduce an enhanced pairwise search approach for generating pairwise test data in terms of optimum in size and least time consumption.

The paper was organized in the way that the detail related work is described in the section 1.1 followed by the proposed algorithm details, the experimental results with discussions and comparison and finally the conclusion suggests for the future work.

1.1 Related Work On Pairwise Test Data Genertion

Empirical facts shows that one of the major sources of software and systems bug/errors [7,8] is the lack of testing for both functional and nonfunctional. NIST (National Institute of Standard and Technology) have shown that an estimated cost of $ 6X 10^{10} wastes as a result of software failure, which was the 0.6 percent of GDP [9, 10]. The research showed that more than one-third of this cost can be reduced by improving software testing mechanism. In this case, automatic testing is of critical concern [11]. For the automation, software can become more practical and scalable. But the automated generation of test data case is challenging [12]. Underlying problem for this is known to be not-decidable and NP-hard, so researchers have focused on the techniques that search to find near optimal test data sets in a reasonable time [13, 14].

A very significant and promising approach called Pairwise testing approach to software testing is used as it often provides efficient error detection at a very low cost. It keeps a well balance between the magnitude and effectiveness of combinations. This requires every combination of any two parameter values to be covered by at least one test case [14,15]. In the pairwise approach there are some pre-defined rules to calculate the test cases data directly from the mathematical functions; those are known as algebraic strategy [4]. On other hand, computational approaches are based on the calculation of coverage of generated pairs, followed by an iterative or random search technique for creating test cases.

Computational approach is used in the IRPS algorithm [6]. This approach is a deterministic strategy that generates all pairs and stores it into the linked list. At the end it searches the entire list, from there it selects the best list and empties the list. In this way when all list become empty, the collection of best list is determined and set as the final test data suite.

By using computational approach, Automatic Efficient Test Generator (AETG) [16] and its divergence mAETG [6], generates pairwise test data. In this approach it uses the 'Greedy technique' to build test cases based on covering as much as possible un-covered pairs. A random search algorithm is used in AETG [16]. Ant Colony Algorithm (ACA) and Genetic Algorithm (GA) are the variants of AETG [16]. Genetic Algorithm [17] creates an initial population of individuals (test cases), then the fitness of those individuals are calculated. After this it starts discarding the unfit individuals by individual selection methods. The genetic operators such as crossover and mutations are applied on the selected individuals. This process continues until a set of best individuals found. In Ant Colony Algorithm [17] the candidate solution is associated with the start and end points. If an Ant chooses one edge from different edges, it would choose the edge with a large amount of pheromone, which gives better result with the higher probability.

In case of In-Parameter-Order (IPO) [12] strategy, it starts with an empty test set and adds one test parameter at a time for pairwise testing. By combination of the first two parameter it creates the test cases, then it adds third and calculate how many pair has been covered. This way it goes on until all the values of each parameter is checked. This is a deterministic approach.

A popular algorithm called AllPairs [18], can generate test suites covering all pairwise interactions within a reasonable time. This strategy looks like to be a deterministic strategy because the same test suit are generated every run time.

A deterministic strategy called the Simulated Annealing (SA) [22] algorithm with the same generated test suite for every run time.

Another algorithm called G2Way (Generalization of Two Way test data) is one of the excellent tools based on computational and deterministic strategy. This strategy based on backtracking algorithm and uses customized markup language to describe base data. This G2Way backtracking algorithm tries to combine generated pairs so that it covers highest pairs. Finally covering all the pairs, the test case treats as final test data suite.

1.2 Our Proposed Strategy

Our proposed algorithm works as follows: Initially, it creates pair parameters and their values. Values of one pair is combined with another pair by calculating the highest possible coverable pairs. This way it constructs a test case and adds to the final test suit. Let's have an example for easy understanding of the algorithm. The scenario is presented as follows:

In Table 1, there are 6 parameters A, B, C, D, E, and F, each of them having 2 values (i.e. a1, a2, b1, b2 etc.). The proposed algorithm first generates pair parameters which are AB, CD, EF and then generate the exhaustive test cases as shown in Table 2.

According to the algorithm every AB pair tries to combine with one CD pairs. If the combined pairs give highest coverage or maximum coverage, then it tries to combine with the next available values (i.e EF pairs). In this case the test case generation approach is in greedy manner and constructs test cases one at a time.

From Table 2, proposed algorithm tries to combine [a1, b1] with 4 possible values of CD in the list [[c1, d1], [c1, d2], [c2, d1], [c2, d2]]. We can see that the first pair which gives the highest coverage looks for the available pairs, which is [[e1, f1], [e1, f2], [e2, f1], [e2, f2]] as shown in Table 3. The highest coverage will then added to the final test suit.

Table 3 shows, one of AB pairs [a1, b1] searches for the best pairs among the available pairs of CD and the output from this should be only one, which is [a1, b1, c1, d1] as the first uncovered final test-case with full coverage. Again, generated pair [a1, b1, c1, d1] search for the available pairs of EF and the generated output is [a1, b1, c1, d1, e1, f1]. This same procedure is followed by other AB pair parameters to generate other final test cases from the list. This way the highest coverable pairs are stored on the final test set. Figure 1 shows the test cases generation algorithm and Figure 2 shows the corresponding flowchart.

1.3 Experimental Results

To evaluate the efficiency of our proposed algorithm, for pairwise test data generation, we have considered 5 different system's configurations. Among these systems the first 3 system are uniform parameterized values and the rest systems are non-uniform show as follows:

S1: 3 3-valued parameters,
S2: 4 3-valued parameters,
S3: 13 3-valued parameters,
S4: 10 5-valued parameters,
S5: 1 5-valued parameters, 8 3-valued parameters and 2 2-valued parameters.

Consideration of the parameters and assumptions are taken according to some of the related existing algorithms that support pairwise test case generation to compare our results with those.

Table 4 shows the comparison of generated test suite size by our proposed algorithm with others. The shadowed cells show the best performance in term of generated test case size. It shows that our proposed algorithm produces the best results in S1, S2, and S4 (shaded) but not for S3 and S5. Since the test case production is a NP-complete problem hence it is well known that no strategy may perform the best for all cases. It shows best performances in three cases, which is highest among all related algorithms.

For a good comparison of execution time (i.e., complexity) for all related test strategies, either computing environment should be equal (normally this will not be possible) or need the source code of the method (which is also not available in most of the cases). All Pairs tool [18] which is free to download and can execute using any computing platform. Hence we have managed to compare the execution time of R2Way with our proposed method using the same platform as follows: Intel Dual Core 2.66 GHz, 1 GB RAM, Java programming language, and OS environment was Windows XP.

1.4 Conclusion

In this paper we have proposed pair parameter based search algorithm which will generate test cases for pairwise testing. The suitability of the proposed algorithm is noticeable. The proposed system combines two parameters together (a single pair) and searches for another pair. In this system the architecture and the algorithm is far different than other existing algorithms because at first the parameters create the pair among themselves and all the pairs look for other pairs to obtain the highest coverage. The main strategy of this algorithm is to generate test cases from the parameter pairs. We have shown the design and description of the algorithm has been shown in other sections. At the end the experimental result shows the efficiency of the algorithm. We have shown that the proposed algorithm is efficient in terms of execution time and able to generate much reduced test suites to fulfill the current demand by the software development companies. The proposed algorithms could be further extended to support higher t-way interaction testing where there are scopes for further research.

References

[1] Xiang Chen, Qing Gu, Jingxian Qi, Daoxu Chen, "Applying Particle Swarm optimization to Pairwise Testing", in proceedings of the 34th Annual IEEE Computer Software And Application Conference, Seoul, Korea, 2010.

[2] Yingxia Cui, Longshu Li, Sheng Yao, "A New strategy for pairwise test case generation", in proceedings of the third international Symposium on Intelligent Information Technology Application, NanChang, China, 2009.

[3] Y. Lei, R. Kacker, D.R. Kuhn, V. Okun, and J.Lawrence, "IPOG: A general strategy for t-way software testing", in proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, Tucson, Arizona, 2007.

[4] M. I. Younis, K. Z. Zamli, N. A. Mat Isa, "Algebraic Strategy to Generate Pairwise Test Set for Prime Number Parameters and Variables", in proceedings of the IEEE international conference on computer and information technology, Kuala Lumpur, Malaysia, 2008.

[5] Mohammad F. J. Klaib, Sangeetha Muthuraman, Noraziah Ahmad, and Roslina Sidek, "A Tree Based Strategy for Test Data Generation and Cost Calculation for Uniform and Non-Uniform Parametric Values", in proceedings of the 10th IEEE international conference on computer and information technology, West Yorkshire, UK, 2010.

[6] M. I. Younis, K. Z. Zamli, N. A. Mat Isa, "IRPS - An Efficient Test Data Generation Strategy for Pairwise Testing," in Proceedings of the 12th international conference on Knowledge-Based Intelligent Information and Engineering Systems, Lecture Notes In Artificial Intelligence, Springer-Verlag, 2008.

[7] D. Leffingwell and D. Widrig, "Managing Software Requirements: A Use Case Approach", Addison Wesley, 2003.

[8] R. L. Glass, "Facts and Fallacies of Software Engineering", Addison Wesley, 2002.

[9] National Institute of Standards and Technology, "The Economic Impacts of Inadequate Infrastructure for Software Testing," Planning Report, 2-3 May, 2002.

[10] Mark Harman and Phil McMinn, "A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search", IEEE Transactions on Software Engineering, vol. 36, no. 2, pp-226-247, 2010.

[11] P. McMinn, "Search-Based Software Test Data Generation: A Survey," Software Testing, Verification and Reliability, vol. 14, no. 2, pp. 105-156, 2004.

[12] Y. Lei and K. C. Tai, "In-Parameter-Order: A Test Generation Strategy for Pairwise Testing", in proceedings of the 3rd IEEE International conference on High-Assurance Systems Engineering, Washington, DC, USA, 1998.

[13] D. Gong, X. Yao, "Automatic detection of infeasible paths in software testing" IET Software, vol. 4, no. 5, pp-361-370, 2010.

[14] Jangbok Kim, Kyunghee Choi, Daniel M. Hoffman, Gihyun Jung, "White Box Pairwise Test Case Generation", in proceedings of the IEEE Seventh International Conference on Quality Software, Oregon, USA, 2007.

[15] Zainal Hisham Che Soh, Syahrul Afzal Che Abdullah, Kamal Zuhari Zamli, "A Parallelization Strategies of Test Suites Generation for t-way Combinatorial Interaction Testing", in proceedings of the IEEE International conference on Information Technology, International Symposium , Kuala Lumpur, Malaysia, 2008.

[16] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design," IEEE Transactions on Software Engineering, vol. 23, no. 7, pp. 437-444, 1997.

[17] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing," in proceedings of the 28th Annual Int. Computer Software and Applications Conf. (COMPSAC'04), Hong Kong, 2004.

[18] J. Bach, "Allpairs Test Case Generation Tool", Available from: http://tejasconsulting.com/open-testware/feature/allpairs.html (Last access date: 27th Sep. 2009)

[19] "TConfig," Available from: http://www.site.uottawa.ca/~awilliam/. (Last access date: 27th Sep. 2009)

[20] M. Harman and B.F. Jones, "Search-based Software Engineering & Information and Software Technology", pp. 833-839, 2001.

[21] Xiang Chen, Qing Gu, Xin Zhang, Daoxu Chen, "Building Prioritized Pairwise Interaction Test Suites with Ant Colony Optimization", in proceedings of the 9th International IEEE Conference on Quality Software, Jeju, Koria, 2009.

[22] James D. McCaffrey, "Generation of Pairwise Test Sets using a Simulated Bee Colony Algorithm", in proceedings of the IEEE International Conference on, Information Reuse & Integration, Las Vegas, USA, 2009.

[23] M. F. J. Klaib, K. Z. Zamli, N. A. M. Isa, M. I. Younis, and R. Abdullah, "G2Way – A Backtracking Strategy for Pairwise Test Data Generation," in proceedings of the 15th IEEE Asia-Pacific Software Engineering Conf, Beijing, China, 2008.

Table 1: Example parameters with values

Parameters	**A**	**B**	**C**	**D**	**E**	**F**
Values	a1	b1	c1	d1	e1	f1
	a2	b2	c2	d2	e2	f2

Table 2: Pair parameters with exhaustive test cases

Pair Parameters	**AB**	**CD**	**EF**
Generated all possible test cases	[a1, b1]	[c1, d1]	[e1, f1]
	[a1, b2]	[c1, d2]	[e1, f2]
	[a2, b1]	[c2, d1]	[e2, f1]
	[a2, b2]	[c2, d2]	[e2, f2]

Table 3: Example of pair search and final test case generation

Initial pairs	**Available pairs**	**Best uncovered pairs**	**Available Pairs**	**Best uncovered pairs**
[a1, b1]	[c1, d1]	[a1, b1, c1, d1]	[e1, f1]	[a1, b1, c1, d1, e1, f1]
	[c1, d2]		[e1, f2]	
	[c2, d1]		[e2, f1]	
	[c2, d2]		[e2, f2]	
[a1, b2]	[c1, d1]	[a1, b2, c1, d2]	[e1, f1]	[a1, b2, c1, d2, e1, f2]
	[c1, d2]		[e1, f2]	
	[c2, d1]		[e2, f1]	
	[c2, d2]		[e2, f2]	
[a2, b1]	[c1, d1]	[a2, b1, c2, d1]	[e1, f1]	[a2, b1, c2, d1, e2, f1]
	[c1, d2]		[e1, f2]	
	[c2, d1]		[e2, f1]	
	[c2, d2]		[e2, f2]	
[a2, b2]	[c1, d1]	[a2, b2, c2, d2]	[e1, f1]	[a2, b2, c2, d2, e2, f2]
	[c1, d2]		[e1, f2]	
	[c2, d1]		[e2, f1]	
	[c2, d2]		[e2, f2]	

Algorithm to Generate Test Suits ()

```
# This algorithm was implemented with C language#
Begin
Let $P_P$ = {} represents the set of all possible pairs
Let $P_S$ = {} represents the pairs where all the $P_S$ stores in $P_P$
Let $P_B$ = {} represents the best pairs set which cover highest pairs
Let $P_F$ = {} as empty set represents the Final test suits
Let $C_B$ as number = 0 represents the best covering number
Let $C_C$ as number = 0 represents the current covering number
For each $P_S$ as $P_1$ in $P_P$
  For each next $P_S$ as $P_2$ in $P_P$
      Add $P_1$ with $P_2$ and put in P
      $C_C$ = Get coverage pair number of P
      IF $C_B$ is less or equal to $C_C$
          Put $C_C$ into $C_B$
          Put $P_2$ into $P_B$
      End IF
  End For
Add $P_1$ with $P_B$ and store to $P_F$
End For
End
```

Figure 1: Proposed Algorithm pseudo code for test case generation.

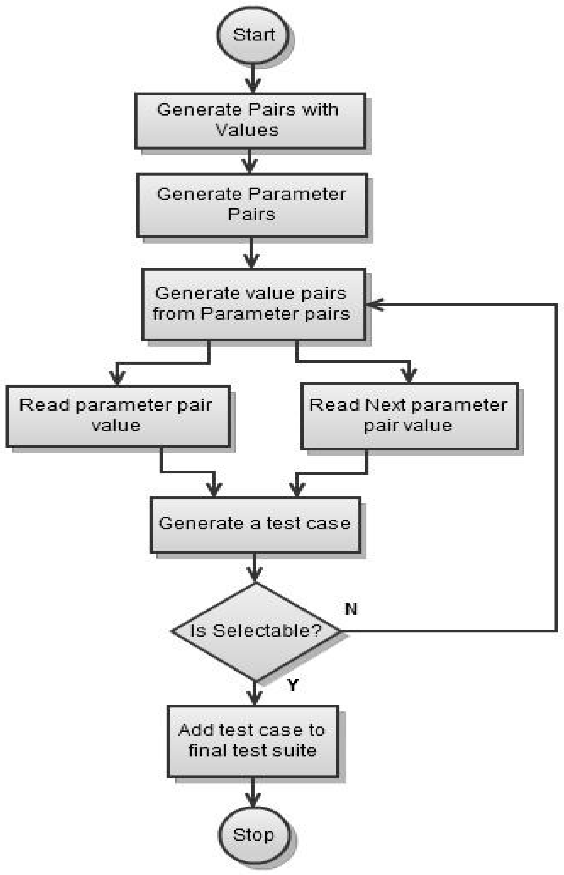
Start
Generate Pairs with Values
Generate Parameter Pairs
Generate value pairs from Parameter pairs
Read parameter pair value
Read Next parameter pair value
Generate a test case
Is Selectable?
N
Y
Add test case to final test suite
Stop

Figure 2: Flow chart of the proposed Algorithm

Table 4: Comparison based on the generated test size

Sys	AETG [15]	AETGm [6]	IPO [18]	SA [26]	GA [15]	ACA [15]	ALL Pairs [16]	G2Way [17]	Proposed system
S1	NA	NA	NA	NA	NA	NA	11	11	11
S2	8	10	8	8	8	8	10	10	8
S3	15	17	17	16	17	17	22	19	22
S4	NA	NA	47	NA	NA	NA	49	43	42
S5	17	19	NA	14	14	16	21	22	22

Table 5: Comparison Based on Execution Time (in seconds)

Sys	**ALL Pairs [18]**	**Proposed System**
S1	0.07	0.017
S2	0.23	0.07
S3	0.40	0.15
S4	0.92	1.22
S5	0.35	0.38