IRPS – An Efficient Test Data Generation Strategy for Pairwise Testing

Mohammed I. Younis, Kamal Zuhairi Zamli, and Nor Ashidi Mat Isa

School of Electrical and Electronic Engineering, Universiti Sains Malaysia,
14300 Nibong Tebal, Penang, Malaysia
{eekamal,ashidi}@eng.usm.my

Abstract. Software testing is an integral part of software engineering. Lack of testing often leads to disastrous consequences including loss of data, fortunes, and even lives. In order to ensure software reliability, many combinations of possible input parameters, hardware/software environments, and system configurations need to be tested and verified against for conformance. Due to costing factors as well as time to market constraints, considering all exhaustive test possibilities would be infeasible (i.e. due to combinatorial explosion problem). Earlier work suggests that pairwise sampling strategy (i.e. based on two-way parameter interaction) can be effective. Building and complementing earlier work, this paper discusses an efficient pairwise test data generation strategy, called IRPS. In doing so, IRPS is compared against existing strategies including AETG and its variations, IPO, SA, GA, ACA, and All Pairs. Empirical results demonstrate that IRPS strategy, in most cases, outperformed other strategies as far as the number of test data generated within reasonable time.

1 Introduction

Software testing is an integral part of software engineering. Lack of testing often leads to disastrous consequences including loss of data, fortunes, and even lives. To ensure acceptable quality and reliability, many combinations of possible input parameters, hardware/software environments, and system configurations need to be considered and verified against for conformance. This consideration often leads to combinatorial explosion problem. Given limited time and resources, it is often impossible to exhaustively consider all of these combinations. Thus, a sampling strategy is needed to select a subset of these combinations in a systematic manner.

Earlier work suggests that pairwise sampling strategy (i.e. based on two-way parameter interaction) can be effective to uncover between 60 to 80 percent of faults [9] [10]. Here, any two combinations of parameter values are to be covered by at least one test [2]. Building and complementing earlier work, this paper proposes and implements an efficient pairwise test data generation strategy, called IRPS. In doing so, IRPS is compared against existing strategies consisting of AETG [2] and its variations [4], IPO [12], SA [15], GA [15], ACA [15], and All Pairs [16]. Empirical results demonstrate that IRPS strategy, in most cases, outperformed other strategies as far as the number of test data generated within reasonable time.

I. Lovrek, R.J. Howlett, and L.C. Jain (Eds.): KES 2008, Part I, LNAI 5177, pp. 493–500, 2008.

2 Related Work

Existing strategies can be categorized into two dominant approaches, that is, algebraic approaches or computational approaches [10].

Algebraic approaches construct test sets using pre-defined rules. Most algebraic approaches compute test sets directly by a mathematical function [10]. Thus, the computations involved in algebraic approaches are typically lightweight, and in some cases, algebraic approaches can produce the most optimal test sets. However, algebraic approaches often impose restrictions on the system configurations to which they can be applied [10] [18]. In a nut shell, algebraic approaches are often based on the extensions of the mathematical methods for constructing orthogonal arrays (OA) [1] [14], and covering arrays (CA) [8] [19]. Some variations of the algebraic approach also exploit recursion in order to permit the construction of larger test sets from smaller ones (see reference [17]).

Unlike algebraic approaches, computational approaches often rely on the generation of the all pair combinations. Based on all pair combinations, the computational approaches iteratively search the combinations space to generate the required test case until all pairs have been covered. Unlike algebraic approaches, the computational approaches can be applied to arbitrary system configurations. Nevertheless, in the case where the number of pairs to be considered is significantly large, adopting computational approaches can be expensive due to the need to consider explicit enumeration from all the combination space.

Adopting the computational approaches as the main basis, an Automatic Efficient Test Generator (or AETG) [2] and its variant (AETG2), employs a greedy algorithm to construct the test case, that is, each test covers as many uncovered combinations as possible. Because AETG uses random search algorithm, the generated test case is highly non-deterministic (i.e. the same input parameter model may lead to different test suites [7]). Other variants to AETG that use stochastic greedy algorithms are: GA (Genetic Algorithm) and ACA (Ant Colony Algorithm) [15]. In some cases, they give optimal solution than original AETG, although they share the common characteristic as far as being non-deterministic in nature.

In Parameter Order (IPO) strategy [11][12], builds a pairwise test set for the first two parameters. Then, IPO strategy extends the test set to cover the first three parameters, and continues to extend the test set until it builds a pairwise test set for all the parameters. In this manner, IPO generates the test case with greedy algorithms similar to AETG. Nevertheless, apart from deterministic in nature, covering one parameter at a time allows the IPO strategy to achieve a lower order of complexity than AETG. All Pairs strategy (i.e. downloadable tool) appears to share the same property as far as producing deterministic test cases is concerned although little is known about the actual strategies employed due to limited availability of references [16][6].

As far as other non-greedy strategies are concerned, some approaches opted to adopt heuristic search techniques such as hill climbing and simulated annealing (SA) [18]. Briefly, hill climbing and simulated annealing strategies start from some known test set. Then, a series of transformations were applied (starting from the known test set) until an optimum set is reached to cover all the pairwise combinations [18]. Unlike AETG and IPO, which builds a test set from scratch, heuristic search techniques can predict the known test set in advance. As such, heuristic search techniques

can produce smaller test sets than AETG and IPO, but they typically take longer time to complete [10].

3 The Proposed Strategy

Strategizing to construct minimum test set from the exhaustive test space is a NP-complete problem [11], that is, it is often unlikely that efficient strategy exists that can lways generate optimal test set (i.e. each interaction pair is covered by only one test). Additionally, the size of the minimum pair wise test set also grows logarithmically with the number of parameter and quadratically with the number of values [2]. Motivated by such a challenge, we have opted to develop IRPS as a research vehicle to investigate efficient strategy and data structure implementation to generate optimal pairwise test set that can eventually be generalized for higher order interactions. Adopting the computational approaches as its basis, the IRPS strategy for generating pairwise test data set takes the following steps:

- Step 1: Generates all pairs and store them into compact linked list called Pi.
- Step 2: Search the Pi list and take the desired weight of the candidate case as a test case then delete it from the Pi list.
- Step 3: repeat step 2 until the Pi list is empty.

As indicated above, the generated pairs are stored in compact linked list called Pi, which is a linked list of linked lists. For a test set with N parameters, the Pi list contains (N-1) linked list. Each linked list contains nodes equal to the number of values defined by its parameter as well as an array of linked list that represents the pair of all other variables in the next linked lists.

To understand how the Pi list works, consider a 4 3-valued parameters system, A = {a0,a1,a2}; B = {b0,b1,b2}, C = {c0,c1,c2}, and D {d0,d1,d2}. In this example, we have $\binom{4}{2}3^2 = 54$ possible pairs of combinations.

Table 1. Pi Linked list for storing combination pairs for 4-3 valued parameters

(index) i= 0	i= 1	i= 2
a 0	b 0	c 0
b 0 b 1 b 2	c 0 c 1 c 2	d 0 d 1 d 2
c 0 c 1 c 2	d 0 d 1 d 2	
d 0 d 1 d 2		
a 1	b 1	c 1
b 0 b 1 b 2	c 0 c 1 c 2	d 0 d 1 d 2
c 0 c 1 c 2	d 0 d 1 d 2	
d 0 d 1 d 2		
a 2	b 2	c 2
b 0 b 1 b 2	c 0 c 1 c 2	d 0 d 1 d 2
c 0 c 1 c 2	d 0 d 1 d 2	
d 0 d 1 d 2		

In this case, the complete Pi linked list can be visualized as in Table 1 given earlier. Node a0 with the pairs linked list array contains the following pairs (<a0,b0>, <a0,b1>, <a0,b2>, ……,<a0,d2>). Here, this list contains only pairs that are based on a0. Similarly, the same observation can be seen with other nodes in the lists. The significant of such arrangement is the fact that less storage unit is required as compared to storing all pairs in clear pairwise combinations. Considering the aforementioned example and assuming each variable takes a unit of storage, then arranging in clear pairwise combinations would require (54*2=108) storage unit. Using similar calculation, adopting our arrangement strategy requires merely 3+(3*9)+3+(3*6)+ 3+(3*3)=33 storage unit.

To describe the IRPS strategy in details, it is necessary to define a number of terminologies. The *weight* of the candidate test case is defined as the number of pairs that are covered by that candidate. For example, the test case combination of a0b0c0d0 covers the pairs (<a0,b0>,<a0,c0>,<a0,d0>,<b0,c0>,<b0,d0>, and <c0,d0>) and the variables b0,c0,d0 in node a0, c0,d0 in node b0, and finally d0 in node c0 , so its weight=6. *The maximum weight*, wmax, for N parameters can be calculated by the following:

$$wmax= N*(N-1)/2$$

Here, if N=4, then wmax=4*3/2=6. *The miss variable* is defined as the difference between the maximum weight and the weight of the candidate test case. *The intersection of node* in the list i with the list (i+1) is defined as the intersection between the node and all nodes given by the first row. IRPS strategy constructs a double linked list that stores the original i node and the intersection with the second node in i+1 list, as well as the rest of the nodes. If the first row in the pairs array is empty, the intersection process will be performed with all values of the nodes in the next list and the miss variable is reduced by one (if miss>0). Otherwise, the intersection process will be terminated and the iteration moves to the next node. *The candidate test* case is obtained by taking the node value in each node in the doubly linked list. For the last node, the candidate test case takes the current value and the first element in the pair array. The candidate test case is taken as a test case only if its weight satisfies the desired weight criteria. If not, the intersection process will continue with the other nodes in the list (by deleting the last node in the doubly linked list and replace it with

```
for (i=0;i<N-;i++) // i is the index of pi list
 begin    //start the search with maximum weight
  w=N(N-1)/2;
  while (list(i) is not empty
   begin
     if (there exist candidate test case  from the intersection of  a node in ith List
        with the remaining i+1 ,…,N-1 Lists)
       delete the test case from pi list;
    else //not find a test case with the desired weight so :
       w--; //decrease the weight
  end
end
```

Fig. 1. The search algorithm

the intersection with next node in the list, or when there is no next node in the list, the strategy will delete the last two nodes and continue with the iteration). In other words, the intersection process goes horizontally when the target weight is not found and grows vertically in recursive fashion. Finally, the *delete operation* operates by deleting each variable (if they exist) in each node.

Figure 1 depicts the search algorithm for the proposed IRPS strategy. Here, the algorithm is terminated whenever the Pi list is empty in order to guarantee that all pairs are covered and each pair only appears at most once in the final generated test cases (i.e. to achieve optimum solution).

4 Evaluation

Our evaluation has two main goals. Firstly, we want to investigate the growth in the size of the test sets generated by IRPS strategy, as well as the time taken to produce those test sets based on the given number of parameters and values. Secondly, we want to compare the performance of IRPS against existing tools particularly in terms of the size and the time taken to produce the test sets. To perform the evaluation, we have applied IRPS to three series of system configurations. In the first series, the number of parameters (p) and the number of variables (v) are equal to each other, the numbers(n) are (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 and 16) respectively. In the second series, the number of parameters is fixed to be 5, and the number of variables is varied from 2 to 10.

Table 2. Results for n=2 to 11 n n-valued parameters

Case Name	CA1	CA2	CA3	CA4	CA5	CA6	CA7	CA8	CA9	CA10
n=p=v	2	3	4	5	6	7	8	9	10	11
size	4	9	16	25	44	49	64	116	149	121
time	<0.001	<0.001	0.011	0.015	0.087	0.034	0.077	240.2	16.35	0.121

Table 3. Results for 5 parameters with 2 to 10 values

Case Name	CA11	CA12	CA13	CA14	CA15	CA16	CA17	CA18	CA19
value(v)	2	3	4	5	6	7	8	9	10
size	6	12	16	25	44	49	78	96	114
time	0.01	0.015	0.016	0.015	0.077	0.057	0.133	0.178	0.184

Table 4. Results for 2 to 10 parameters with 5 values

Case Name	CA20	CA21	CA22	CA23	CA24	CA25	CA26	CA27	CA28
parameter(p)	2	3	4	5	6	7	8	9	10
size	25	25	25	25	25	37	41	44	45
time	0.053	0.054	0.114	0.015	0.031	0.32	0.78	1.45	1.928

Tables 2, 3 and 4 show the experimental results for the three series of system configurations respectively. The columns in the three tables are self-explanatory. Note that the execution times are shown in seconds, and all the results were collected using a laptop running Windows Vista with 1.6GHZ CPU and 512 MB memory. The entire tool is implemented using Java Development Kit 1.4 (JDK1.4) platforms.

For pairwise interaction, the optimal size can be viewed as the product of the two maximum numbers of variables. This observation can be seen in the case of CA1, CA2, CA3, CA4, CA6, CA7, and CA10 from Table 2. Similar observation can be seen in the case of CA13, CA14, and CA16 from Table 3. The generated test case is also minimal in size, as depicted in CA20, CA21, CA22, CA23 and CA24 from Table 4 respectively. Here, we conclude that the size of generated test case depends linearly on the optimal size of the generated test case.

As far as execution time is concerned, we observe that the execution time is significantly independent on the number of parameters and values when the size is not minimal. This is due to the nature of the algorithm that generates the heavy weighted test case first, deletes them from the Pi list, and then searches again for the uncovered pairs. In this way, the size of the generated test case and the execution time depend on the phenomena of greedy algorithm rather than the number of parameters and values.

We observe that the size and execution time of CA9 (10 10-valued parameters) is greater than CA10 (11 11-valued parameters), according to Table 2, and the size of CA7 (8 8-valued parameters) is greater than CA17 (8 5-valued parameters) according to Tables 2, and 3 respectively. Here, we conclude that the behavior of IRPS is unpredictable in term of the execution time due to the exhaustive search nature when drifting from optimal size, but running the test case generator produces the same test set on every case (thus, IRPS strategy is deterministic).

As for comparison, we have identified the following existing strategies that support pairwise testing: AETG [2] [3], AETG2 [15] [5], IPO [12], SA [15], GA [15], ACA [15], and All Pairs tool [16]. We consider eight systems namely; S1: 3 3-valued parameters, S2: 4 3-valued parameters, S3: 13 3-valued parameters, S4: 10 10-valued parameters, S5: 10 15-valued parameters, S6: 20 10-valued parameters, S7: 10 5-valued parameters, and S8: 1 5-valued parameters, 8 3-valued parameters and 2 2-valued parameters. The system configurations are: AETG2 & SA: C++, Linux, Intel P IV 1.8 GHZ; IPO: Java, Windows 98, Intel P II 450 MHZ; CA, & ACA: C, Windows XP, P IV 2.26 GHZ; AllPairs: Perl, Windows Vista, P IV 1.6 GHZ, 512 MB RAM; and IRPS: Java, Windows Vista, P IV 1.6 GHZ, 512 MB RAM.

Table 5 shows the size of the test set generated by each strategy, and Table 6 shows the execution time for each system. All the problem instances and data for the existing strategies are taken from [12], [15], and [5] except for All Pairs tool (available freely, which we run side by side with our tool). Entries marked with NA are data that are not available in these papers.

Referring to Table 5, IRPS always generate smaller test cases than ALL Pairs and in some cases generates less (i.e. S4, S5, S6, and S7) or equals to that of IPO (i.e.S2, S3). IRPS also generates less the cases compared to AETG2 (except S6), GA and ACA (except S8). While IRPS outperformed AETG in S8, AETG outperformed IRPS in S3, and S6. Finally, SA outperformed IRPS (in S3, S6, and S8). Unlike AETG, AETG2, GA, ACA and IRPS; SA does not have the practical advantage of the

Table 5. Comparison on the size of the test set generated by existing strategies

System	AETG	AETG2	IPO	SA	GA	ACA	All Pairs	IRPS
S1	NA	NA	NA	NA	NA	NA	10	9
S2	9	11	9	9	9	9	10	9
S3	15	17	17	16	17	17	22	17
S4	NA	NA	169	NA	157	159	177	149
S5	NA	NA	361	NA	NA	NA	390	321
S6	180	198	212	183	227	225	230	210
S7	NA	NA	47	NA	NA	NA	49	45
S8	19	20	NA	15	15	16	21	17

Table 6. Comparison on the time taken to generate test set (in seconds) for existing strategies

System	AETG	AETG2	IPO	SA	GA	ACA	All Pairs	IRPS
S1	NA	NA	NA	NA	NA	NA	0.08	<0.001
S2	NA	NA	NA	NA	NA	NA	0.23	0.004
S3	NA	NA	NA	NA	NA	NA	0.45	39.23
S4	NA	NA	0.3	NA	866	1180	5.03	16.35
S5	NA	NA	0.72	NA	NA	NA	10.36	1124
S6	NA	6001	NA	10833	6365	7083	23.3	3213
S7	NA	NA	0.05	NA	NA	NA	1.02	1.928
S8	NA	58	NA	214	22	31	0.35	2.02

greedy algorithm as the implementation is not based on such an algorithm. Here, in the absence of the greedy algorithm, the construction of the test set can not utilize the useful property that the test case created earlier has more significant impact as far as the interaction coverage is concerned [15].

Admittedly, no fair comparison can be made in terms of execution time from existing strategies due to the differences in the computing environments, and the unavailability of the open source code or executable code to run in our platform (with the exception of ALL Pairs tool). Nevertheless, as a general observation; we believe that the execution time for IRPS is still acceptable as compared to other strategies (see Table 6). Not considering the computing differences, IPO outperforms all other strategies. One reason may be that IPO employs deterministic algorithm and needs only one run. Thus, IPO requires much less time to execute than others. SA includes the time taken to find all sized test sets through binary search process, hence, requiring more run time than others. In short, no strategies can clearly be dominant in all.

To conclude, here in this paper, we propose a novel deterministic computational strategy for pairwise testing with efficient data structure for storing and searching pairs. Our initial evaluation results are encouraging particularly in terms of test suite size within acceptable execution time. As part as our future work, we are currently investigating a new parallel search algorithm for IRPS to be implemented under the GRID environment, supported by the USM GRID - Research University Grant.

References

1. Bush, K.A.: Orthogonal Arrays of Index Unity. Annals of Mathematical Statistics 23, 426–434 (1952)
2. Cohen, D.M., Dalal, S.R., Fredman, M.L., Patton, G.C.: The AETG system: An Approach to Testing Based on Combinatorial Design. IEEE Trans. on Software Engineering 23(7), 437–443 (1997)
3. Cohen, D.M., Dalal, S.R., Parelius, J., Patton, G.C.: The Combinatorial Design Approach to Automatic Test Generation. IEEE Software 13(5), 83–88 (1996)
4. Cohen, M.B.: Designing Test Suites for Software Interaction Testing, PhD Thesis, University of Auckland (2004)
5. Cohen, M.B., Gibbons, P.B., Mugridge, W.B., Colbourn, C.J.: Constructing Test Suites for Interaction Testing. In: Proc. of the 25th Intl. Conf. on Software Engineering (ICSE 2003), Dallas USA. IEEE CS Press, Los Alamitos (2003)
6. Copeland, L.: A Practitioner's Guide to Software Test Design. STQE Publishing (2004)
7. Grindal, M., Offutt, J., Andler, S. F.: Combination Testing Strategies: A Survey. In: GMU Technical Report ISE-TR-04-05 (July 2004)
8. Hartman, A., Raskin, L.: Problems and Algorithms for Covering Arrays. Discrete Mathematics 284(1-3), 149–156 (2004)
9. Kuhn, D.R., Wallace, D.R., Gallo, A.M.: Software Fault Interactions and Implications for Software Testing. IEEE Trans. on Software Engineering 30(6), 418–421 (2004)
10. Lei, Y., Kacker, R., Kuhn, D.R., Okun, V., Lawrence, J.: IPOG: A General Strategy for T-Way Software Testing. In: 14th Annual IEEE Intl. Conf. and Workshops on the Engineering of Computer-Based Systems, Tucson, AZ, March 2007, pp. 549–556. IEEE CS Press, Los Alamitos (2007)
11. Lei, Y., Tai, K.C.: In-Parameter-Order: A Test Generating Strategy for Pairwise Testing. In: Proc. 3rd IEEE Intl. Symp. On High Assurance System Engineering, November 1998, pp. 254–261 (1998)
12. Lei, Y., Tai, K.C.: In-Parameter-Order: A Test Generating Strategy for Pairwise Testing. IEEE Transaction on Software Engineering 28(1), 1–3 (2002)
13. Maity, S., Nayak, A., Zaman, M., Bansal, N., Srivastava, A.: An Improved Test Generation Strategy for Pair-Wise Testing. In: Fast Abstract ISSRE 2003 (2003)
14. Mandl, R.: Orthogonal Latin squares: an application of experiment design to compiler testing. Communications of the ACM 28(10), 1054–1058 (1985)
15. Shiba, T., Tsuchiya, T., Kikuno, T.: Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing. In: 28th Annual Intl. Computer Software and Applications Conference (COMPSAC 2004), Hong Kong, China, September 2004, pp. 72–77 (2004)
16. `http://www.satisfice.com`
17. Williams, A.W., Probert, R.L.: A Practical Strategy for Testing Pair-Wise Coverage of Network Interfaces. In: Proc. of the 7th Intl. Symp. on Software Reliability Engineering (ISSRE), White Plains, New York (1996)
18. Yan, J., Zhang, J.: Backtracking Algorithms and Search Heuristics to Generate Test Suites for Combinatorial Testing. In: Proc. of the 30th Annual Intl. Computer Software and Applications Conference (COMPSAC 2006), Chicago USA, September 2006, vol. 1, pp. 385–394. IEEE CS Press, Los Alamitos (2006)
19. Zekaoui, L.: Mixed Covering Arrays on Graphs and Tabu Search Algorithms. In: MSc Thesis, Ottawa-Carleton Institute for Computer Science, University of Ottawa, Canada (September 2006)