A Deterministic T-Way Strategy for Test Data Minimization

Kamal Z. Zamli, Mohammed F. J. Klaib, Mohammed I. Younis, and Noor Syarida Jusof
School of Electrical and Electronics
Universiti Sains Malaysia
14300 Nibong Tebal, Penang, Malaysia
Email: eekamal@eng.usm.my

Abstract- **In order to meet market demands for quality software products, software engineers are increasingly under pressure to test more lines of codes. To maintain acceptable test coverage, software engineers need to consider a significantly large number of test cases. Many combinations of possible input parameters, hardware/software environments, and system conditions need to be tested and verified against for conformance based on the system's specification. Often, this results into combinatorial explosion of test cases.**

While earlier work has indicated that pairwise testing (i.e. based on 2-way interaction of variables) can be effective to detect most faults in a typical software system, a counter argument suggests such conclusion cannot be generalized to all software system faults. In some system, faults may also be caused by the interaction of more than two parameters (i.e. t-way). In order to address some of these issues, this paper discusses a novel strategy, called GTWay. Utilizing two main algorithms (i.e. the t-way pair generation algorithm and the backtracking algorithm respectively) as its basis, we demonstrate the efficiency and correctness of GTWay for t-way test data minimization.

***Keywords*- t-way Testing, Software Engineering**

I. INTRODUCTION

As an activity for ensuring quality assurances and improving reliability, software testing is an important part of the software engineering lifecycle. Lack of testing often leads to disastrous consequences including loss of data, fortunes and even lives. For these reasons, many inputs parameters and system conditions need to be tested against the system's specification for conformance. Although desirable, exhaustive software testing is next to impossible due to resources as well as timing constraints.

As illustration, 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 220x56 (i.e. 58,720,256) combinations of test cases to be evaluated. Assuming that it takes 5 minute for one test case, then it would require nearly 559 years for a complete test of the View tab option.

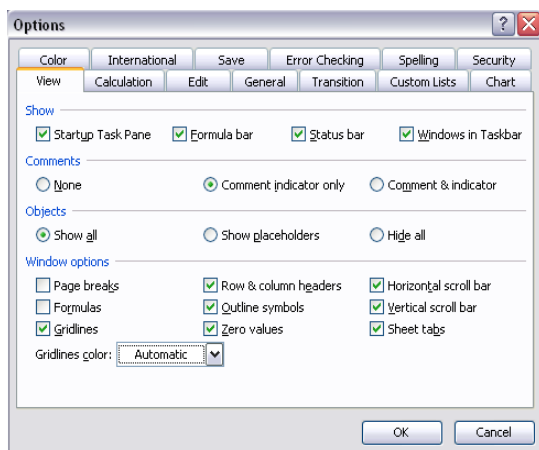

Figure 1. Model of a typical Software System

While earlier work (e.g. in [8, 10]) has indicated that pairwise testing (i.e. based on 2-way interaction of variables) can be effective to detect most faults in a typical software system, a counter argument suggests such conclusion cannot be generalized to all software system faults. For example, the study by The National Institute of Standards and Technology (NIST) [13] reported that 95% of the actual faults on the test software involve 4-way interaction. In fact, almost all of the faults are detected with 6-way interaction. Thus, as this example illustrates, system faults caused by variable interactions may also span more than two parameters (i.e. t-way). Here, t implies the strength of input coverage, that is, t can take up a range of value from $2 \leq t \leq$ the number of parameters, n. If t equals to the number of parameters, n, then t is all full strength (i.e. exhaustive combination). Viewing from a different perspective, t also implies the interaction of parameters. Thus, if $t = n$ is chosen, all n parameter interactions are covered (i.e. exhaustive combinations). Similarly, if $t=n-1$ is chosen, only n-1 interaction of parameters are covered. In this manner, lesser t value always yields lesser combinations.

Considering more than two parameter interaction is not without difficulties. To highlight the difficulties, consider the TCAS is an aircraft collision avoidance system from the Federal Aviation Administration which has been used as case study in other related works [15, 18, 19]. Here, TCAS module has twelve parameters: seven parameters have 2 values, two parameters have three values, one parameter has four values, and two parameters have 10 values. Running exhaustive test requires 460800 (i.e., 10x10x4x3x3x2x2x2x2x2x2x2), or 12 way testing for this system (i.e. running such test may be impossible). Alternatively, 11-way testing requires 230400. 10-way requires 201601. 9-way requires 120361. 8-way requires 56742. 7-way requires 26061. 6-way requires 10851. 5-way requires 4196. 4-way requires 1265. 3-way requires 400. Finally, 2-way requires 100 test cases.

As demonstrated above, when the number of parameter coverage increases, the number of t-way test set also increases exponentially. As such, for large system with many parameters, considering higher order t-way test set can lead toward combinatorial explosion problem. We consider this problem for t-way generation of test set in our design of GTWay. Utilizing two main algorithms (i.e. the t-way pair generation algorithm and the backtracking algorithm respectively), we demonstrate the efficiency and correctness of GTWay for t-way test data minimization.

This paper is structured as follows. Section 2.0 describes some of the related work. Section 3.0 highlights our t-way strategies along with the illustrative example. Section 4.0 discusses the demonstration of correctness of our strategy through interaction coverage analysis. Finally, section 5.0 gives our conclusion.

II. RELATED WORK

A number of strategies have been developed on in the past on t-way testing. Although useful, existing strategies appear to focus more on pairwise (t=2) testing (e.g. AETG [7, 8], AETGm [9], OATS (Orthogonal Array

Test System) [6, 14], G2Way [16], IRPS [23], AllPairs [5], IPO [20], TCG (Test Case Generator) [24], ReduceArray2 [12], DDA (Deterministic Density Algorithm) [11], and OATSGen [17]). Here, we are more interested on a general strategy for t-way test generation for comparative purposes with our work. Thus, what follows is our survey on existing strategies that supports both pairwise and higher order t (i.e. t≥ 2).

IBM's Intelligent Test Case Handler (ITCH) [1] uses combinatorial algorithms based on exhaustive search to construct test suites for t-way testing. Although useful as part of IBM's automated test plan generation, ITCH results is not optimized as far as the number of generated test cases is concerned (i.e. some t-way interaction is covered by more than one test). Additionally, due to its exhaustive search algorithm, ITCH execution typically takes a long time.

Jenny [2], TConfig [3] and TVG [4] are public domain tools (available for download) that support t-way testing. While we are able to execute the tools, their details implementations are not known due to limited information available in the literature. Comparing with TConfig and TVG, Jenny appears to produce less test sets as well as run faster. Additionally, Jenny can support test generation up to t=8. Noted here that we have not been successful to summon Jenny for t >8 as the program implementation crashes.

IPOG [19] is perhaps the most recent strategy for t-way testing. A generalization of a pairwise strategy based on vertical and horizontal extension, IPOG strategy first generates a pairwise test set for the first two parameters. It then continues to extend the test set to generate a pairwise test set for the first three parameters and continues to do so for each additional parameter until all the parameters of the system are covered via horizontal extension. If required (i.e. for interaction coverage), IPOG also employs vertical extension in order to add new tests after the completion of horizontal extension.

As seen above, much useful and significant progress has been achieved as far the support for t-way testing is concerned. Nonetheless, constructing the minimum test set for t-way testing is a NP-complete problem [21, 22] , that is, it is unlikely an efficient algorithm exists that can always generate an optimal test set (i.e. each t-way pair is covered only once [21]). Taking the aforementioned challenges, it is the development of an optimum strategy, called GTWay, for t-way testing is the main focus of this paper.

III. THE GTWAY STRATEGY

As discussed earlier, the GTWay strategy is based on two main algorithms, that is, the t-way pair generation algorithm and the backtracking algorithm respectively. Both of these algorithms will be elaborated next.

A. T-Way Pair Generation Algorithm

In order to facilitate discussion, consider the following running example (with 4 parameters and 2 values). Assume that the interaction strength, t = 3.

TABLE 1
BASE TEST VALUES

Base Value	Input Variables			
	A	**B**	**C**	**D**
	a1	b1	C1	d1
	a2	b2	C2	d2

TABLE 2
INDEX SEARCH FOR A 4 PARAMETER SYSTEM

Index	0	1	2	3	4	5	6	7
Binary	0000	0001	0010	0011	0100	0101	0110	0111
Index	8	9	10	11	12	13	14	15
Binary	1000	1001	1010	1011	1100	1101	1110	1111

TABLE 3
ROW INDEX FOR A 4 PARAMETER SYSTEM

Row Index	**Index**		**Bits**							
			b7	b6	b5	b4	b3	b2	b1	b0
0	7	→	1	1	1	1	1	1	1	1
1	11	→	1	1	1	1	1	1	1	1
2	13	→	1	1	1	1	1	1	1	1
3	14	→	1	1	1	1	1	1	1	1

Here, the pair generation algorithm first identifies all the possible 3 way interactions. Referring to Table 1, the 3 way interactions possibilities are amongst parameters: ABC, ABD, ACD, and BCD. Based on these interactions, the t-way pair generation algorithm generates the following sets:

ABC={(a1,b1,c1), (a1,b1,c2), (a1,b2,c1), (a1,b2,c2), (a2,b1,c1), (a2,b1,c2), (a2,b2,c1), (a2,b2,c2)}

ABD={(a1,b1,d1),(a1,b1,d2), (a1,b2,d1), (a1,b2,d2),(a2,b1,d1), (a2,b1,d2), (a2,b2,d1), (a2,b2,d2)}

ACD={(a1,c1,d1), (a1,c1,d2), (a1,c2,d1), (a1,c2,d2),(a2,c1,d1), (a2,c1,d2), (a2,c2,d1), (a2,c2,d2)}

BCD={(b1,c1,d1), (b1,c1,d2), (b1,c2,d1), (b1,c2,d2),(b2,c1,d1), (b2,c1,d2), (b2,c2,d1), (b2,c2,d2)}

```
Algorithm Pair_Generation (t : t-way value)
1: begin
2:  let Sp ={} as empty set, where Sp represents the pair set
3:  let n∑ = {n0......nm} where n∑ represents the values defined for each parameter,
    m = max no of parameters
4:  let p = {p0 ..pj}, where p represents the sorted set of sets of values defined for each parameter
5:  for index=0 to 2^m - 1
6:   begin
7:     let b = binary number
8:     b = convert index to binary
8:     if (the no of '1's in b = t)
9:      begin
10:      calculate number of possible combinations (PCi )between the partial sets of values
11:      for the shared parameters
12:         begin
13:          multiply {nx x ny} values from n∑
14:          set the bits group (equal to PCi) in the index row to 1
15:         end
16:     end
17:   end
18: end
```

Figure 2. The T-Way Pair Generation Algorithm

```
Algorithm Backtracking (Sp: Set;var St: Set)
1: begin
2:  let St ={} as empty set, where St represents the generated test set
3:  for the first two parameters
4:   begin
5:     create partial the test cases by selecting best values for  higher parameters {P3...Pj},
       that covers the maximum number of uncovered pairwise combinations in Sp
6:     store generated test cases in St
7:     remove covered pairs from Sp (by setting zero values to indicated bits).
8:   end
9:   while still found elements in Sp
10:    begin
11:      add a new element in the St set with empty fields
12:      bring the first uncovered combination, decompose and fills the  initial value in the element set
13:      for the 2nd uncovered combination
14:        begin
15:          decompose uncovered combination
16:          if (current pair element in Sp can be combined with other pair element)
17:            begin
18:              count number of uncovered combination
19:              if (has most uncovered pairs)
20:                fill it in the element set
21:            end
22:          if (the element set does not have matching pair)
23:            select the first element as default values to the missing parameters
24:          store it in St and remove the covered pairs from Sp
25:        end
26:    end
27: end
```

Figure 3. The Backtracking Algorithm

As will be discussed later in the next section, the generated sets serve two purposes. Firstly, any one of these sets can be merged together with another set to form a complete test suite (e.g. ABC and ABD). Secondly, all of the elements in the sets can be used to counter check that all the pairs are indeed covered.

Concerning implementation, the t-way pair generation algorithm initially finds the loop edge for the t-way combinations (i.e. based on the number of defined parameters, P). Then, the algorithm performs index searches through a loop from 0 to 2p -1. Here, for each index, the algorithm converts the number to binary format. Now, if the number of binary one's in the index is equal to t value (i.e. t-way interaction), then that index is put in the index set. Using the same example in Table 1, the loop edge is 15 (i.e. 24 -1). In this case, the index searches loop found 4 indexes having three one's, that is (7, 11, 13, 14) respectively (see Table 2).

In the t-way pair generation algorithm, each index will contain a number of t-way combinations (equals to the multiplication of values defined in each shared parameter). For our example, the first index will have 2x2x2 combinations, the second index will have 2x2x2 combinations, the third index will have 2x2x2 combinations, and forth index will have 2x2x2 combinations. Hence, the total combinations are 32.

To minimize the access time and the space requirements, an efficient data structure (structure of bits) has been designed. Here, row indexes are used to store the indexes of the t-way combinations. Using our example, row index 0 (corresponds to (A, B, C) combinations) stores 8 combinations which are indicated as bits b0 to b7. Similarly, row index 1 stores 8 combinations whilst row index 2 stores 8 combinations and row index 3 stores 8 combinations (see Table 3).

Having described its implementation, the t-way pair generation algorithm can be summarized in Figure 2.

B. The Backtracking Algorithm

The backtracking algorithm randomly chooses the pair sets from the t-way sets and iteratively traverses (or backtracks) each element within the set in order to merge them together to form a complete test suite (see Figure 3). Using our earlier example in Table 1, set ABC, for instance, can be merged with any of the other sets. For illustration sake, assume that the backtracking algorithm randomly chooses the set ABC and ABD for merger. Now, the merger rules works as follows:

- Two elements for each t-way pair sets can only be merged when they are combinable (i.e. the pair elements complement each other's missing value).
- A test case is selected as a result of the merger, only if, it covers the most uncovered t-way pairs. This is to ensure that optimum test suite (i.e. each possible combination is covered at least once) will be generated in the end.

Applying the merger rule whilst traversing the 3-way pair sets of ABC and ABD, the first combinable elements are the first elements of both sets. When the first element in ABC consisting of (a1,b1,c1) is merged with the first element in ABD consisting of (a1,b1,d1), the resulting test case is (a1,b1,c1,d1). Since the test case covers all new 3-way pairs of (a1,b1,c1), (a1,b1,d1), (a1,c1,d1), (b1,c1,d1), the resulting test case is selected as one of the test cases in the final test suite. Upon this selection, the covered pairs are deleted from all of their respective sets. Now, the next combinable element within ABC is (a1,b1,c2) with the second element in ABD consisting of (a1,b1,d2), the resulting test case is (a1,b1,c2,d2). Since the test case covers all new 3-way pairs of (a1,b1,c2), (a1,b1,d2), (a1,c2,d2), (b1,c2,d2), the resulting test case is selected as one of the test cases in the final test suite, and covered pairs are deleted from all of their respective sets. Now, the next combinable element within ABC is (a1,b2, c1) with the third element in ABD consisting of (a1,b2,d1). Here, the resulting test case is (a1,b2,c1,d1).

In this case, the 3-way pairs covered are (a1,b2,c1), (a1,b2,d1), (a1,c1,d1), (b2,c1,d1). Because there exists covered pair of (a1,c1,d1) from earlier merger, the resulting test case of (a1,b2,c1,d1) has not been selected in the final test suite (i.e. not covering the most uncovered 3-way pairs). Now, the traversing and merging process continues until all 3-way pairs are covered. In this example, the final optimum test suite for t=3 is given in Table 4.

Noted here is the fact that all 4 way interaction (i.e. exhaustive) will make up of 24 = 16 combinations (see Table 5).

Considering our running example (i.e. referring to Table 4 and Table 5), there is a minimization of 50% of test suite if the strength is relaxed to 3 instead of 4.

TABLE 4
OPTIMUM ALL 3 WAY INTERACTION

	Input Variables			
	A	**B**	**C**	**D**
Base Values	a1	b1	c1	d1
	a2	b2	c2	d2
	a1	b1	c1	d1
Optimum All 3 Way Solutions	a1	b1	c2	d2
	a1	b2	c1	d2
	a1	b2	c2	d1
	a2	b1	c1	d2
	a2	b1	c2	d1
	a2	b2	c1	d1
	a2	b2	c2	d2

TABLE 5
EXHAUSTIVE COMBINATION

	Input Variables			
	A	**B**	**C**	**D**
Base Values	a1	b1	c1	d1
	a2	b2	c2	d2
All Combinatorial Values	a1	b1	c1	d1
	a1	b1	c1	d2
	a1	b1	c2	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c1	d2
	a1	b2	c2	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c1	d2
	a2	b1	c2	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c1	d2
	a2	b2	c2	d1
	a2	b2	c2	d2

Also, not shown in this example is the case where parameter values are non-uniform (i.e. with unequal number of parameter values). Due to this non-uniformity, some parameter values are considered don't care. In the case, when the pair element (consisting of don't care values) cannot be covered, the backtracking algorithm falls back to the first defined values (i.e. in place of don't care). In this manner, the pair element can still be covered in the final test suite.

IV. DEMONSTRATION OF CORRECTNESS

In order to investigate whether or not the all 3 way pairs are covered, it is necessary to tabulate all the 3 way interaction. As discussed earlier, the interactions are between ABC, ABD, ACD, and BCD. Based on these interactions, the expected total pairs will be 32 (i.e. 8 pairs/interactions x 4 interactions).

Here, we will focus on demonstrating the correctness of the GTWay strategy by analyzing the resulting test suite. Here, we aim to show that GTWay gives optimum results, that is, all 3 way pairs of combinations are covered at most by one test. Based on Table 4, Table 6 lists all the 3 way pairs along with the individual test case generated by GTWay strategy that cover them (denoted as T#).

TABLE 6
FINAL TEST SUITE FOR T=3

Test ID					
	T#1	a1	b1	c1	d1
	T#2	a1	b1	c2	d2
	T#3	a1	b2	c1	d2
	T#4	a1	b2	c2	d1
	T#5	a2	b1	c1	d2
	T#6	a2	b1	c2	d1
	T#7	a2	b2	c1	d1
	T#8	a2	b2	c2	d2

As the pair coverage is 100%, and each of the pair occurrences is only once, we can conclude that GTWay, in this example, produces an optimum test suite. Referring to Table 7, we observe that each combination pair appears at most once (which means that the generated test cases include all generated pairs) and there is no missing pair (which means that our strategy is correct).

TABLE 7
PAIR COVERAGE FOR T=3

Interaction	3 Way pairs	Covered by	Occurences
ABC	a1, b1,c1	T#1	1
	a1,b1,c2	T#2	1
	a1,b2,c1	T#3	1
	a1,b2,c2	T#4	1
	a2, b1,c1	T#5	1
	a2,b1,c2	T#6	1
	a2,b2,c1	T#7	1
	a2,b2,c2	T#8	1
ABD	a1, b1,d1	T#1	1
	a1,b1,d2	T#2	1
	a1,b2,d1	T#3	1
	a1,b2,d2	T#4	1
	a2, b1,d1	T#5	1
	a2,b1,d2	T#6	1
	a2,b2,d1	T#7	1
	a2,b2,d2	T#8	1
BCD	b1,c1,d1	T#1	1
	b1,c1,d2	T#2	1
	b1,c2,d1	T#3	1
	b1,c2,d2	T#4	1
	b2, c1,d1	T#5	1
	b2,c1,d2	T#6	1
	b2,c2,d1	T#7	1
	b2,c2,d2	T#8	1
ACD	a1,c1,d1	T#1	1
	a1,c1,d2	T#2	1
	a1,c2,d1	T#3	1
	a1,c2,d2	T#4	1
	a2, c1,d1	T#5	1
	a2,c1,d2	T#6	1
	a2,c2,d1	T#7	1
	a2,c2,d2	T#8	1

CONCLUDING REMARK

Summing up, this paper has described the development of a deterministic t-way strategy (called GTWay) for systematic software test data minimization. Additionally, this paper has also detailed out the algorithms used for GTWay. Results demonstrate that the algorithms work well in terms of optimizing the coverage of the pairs. More work is still required in order to evaluate the applicability and suitability of GTWay using real test data.

In line with the increasing market demands and deadline for more functionality, test engineers are often under pressure to test more and more code implementations and modules in order to maintain the required level of product quality. With the support of this technique, the effort on testing can be significantly reduced (as the test data set is also reduced). Hence, the product can be out in the market sooner and with less testing overhead.

ACKNOWLEDGEMENT

This research is funded by the eScienceFund grants – "Development of a Mobile Agent Based Parallel and Automated Java Testing Tool".

REFERENCES

1. IBM Intelligent Test Case Handler, Available from: http://www.alphaworks.ibm.com/tech/whitch, 2005.
2. Jenny, Available from: http://www.burtleburtle.net/bob/math/.
3. TConfig, Available from: http://www.site.uottawa.ca/~awilliam/.
4. TVG, Available from: http://sourceforge.net/projects/tvg.
5. Bach, J. Allpairs Test Case Generation Tool, Available from: http://tejasconsulting.com/open-testware/feature/allpairs.html.
6. Bush, K.A. Orthogonal Arrays of Index Unity. The Annals of Mathematical Statistics, 23 (3). 426-434.
7. Cohen, D.M., Dalal, S.R., Fredman, M.L. and Patton, G.C. The AETG System: An Approach to Testing Based on Combinatorial Design. IEEE Transactions on Software Engineering, 23 (7). 437-444.
8. Cohen, D.M., Dalal, S.R., Kajla, A. and Patton, G.C., The Automatic Efficient Test Generator (AETG) System. in Proc. of the 5th International Symposium on Software Reliability Engineering, (Monterey, CA, USA, 1994), 303-309.

9. Cohen, M.B. Designing Test Suites for Software Interaction Testing Computer Science, University of Auckland, New Zealand, September 2004.
10. Cohen, M.B., Gibbons, P.B., Mugridge, W.B. and Colbourn, C.J., Constructing Test Suites for Interaction Testing. in Proc. of the 25th International Conference on Software Engineering, (Portland, Oregon USA, 2003), 38-48.
11. Colbourn, C.J., Cohen, M.B. and Turban, R.C., A Deterministic Density Algorithm for Pairwise Interaction Coverage. in Proc. of the IASTED Intl. Conference on Software Engineering (Innsbruck, Austria, 2004), 345-352.
12. Daich, G.T., Testing Combinations of Parameters Made Easy [Software Testing]. in IEEE Systems Readiness Technology Conference (AUTOTESTCON 2003), (2003), 379-384.
13. Dalal, S.R., Jain, A., Karunanithi, N., Leaton, J.M., Lott, C.M., Patton, G.C. and Horowitz, B.M., Model Based Testing in Practice. in Proc. of the Intl. Conf. on Software Engineering (ICSE), (1999), 285–294.
14. Hedayat, A.S., Sloane, N.J.A. and Stufken, J. Orthogonal Arrays: Theory and Applications. Springer Verlag, New York, 1999.
15. Hutchins, M., Foster, H., Goradia, T. and Ostrand, T., Experiments on the Effectiveness of Dataflow- and Control Flow-Based Test Adequacy Criteria. in Proc. of the 16th International Conference on Software Engineering (ICSE-16), (Sorrento, Italy, May 1994), 191–200.
16. Klaib, M.F.J., Zamli, K.Z., Isa, N.A.M., Younis, M.I. and Abdullah, R., G2Way – A Backtracking Strategy for Pairwise Test Data Generation. in Proc. of the 15th IEEE Asia-Pacific Software Engineering Conf, (Beijing, China, 2008), 463-470.
17. Krishnan, R., Krishna, S.M. and Nandhan, P.S. Combinatorial Testing: Learnings from our Experience. ACM SIGSOFT Software Engineering Notes, 32 (3). 1-8.
18. Kuhn, D.R. and Okum, V., Pseudo-Exhaustive Testing for Software. in the 30th Annual IEEE/NASA Software Engineering Workshop (SEW '06), (April 2006), 25-27.
19. Lei, Y., Kacker, R., Kuhn, D.R., Okun, V. and Lawrence, J., IPOG: A General Strategy for T-Way Software Testing. in Proc. of the 14th Annual IEEE Intl. Conf. and Workshops on the Engineering of Computer-Based Systems, (Tucson, AZ U.S.A, 2007), 549-556.
20. Lei, Y. and Tai, K.C., In-Parameter-Order: A Test Generation Strategy for Pairwise Testing. in Proc. of the 3rd IEEE Intl. High-Assurance Systems Engineering Symp, (Washington, DC, USA, 1998), 254-261.
21. Shiba, T., Tsuchiya, T. and Kikuno, T., Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing. in Proc. of the 28th Annual Intl. Computer Software and Applications Conf. (COMPSAC'04), (Hong Kong, 2004), IEEE Computer Society, 72-77.
22. Tai, K.C. and Lei, Y. A Test Generation Strategy for Pairwise Testing. IEEE Transactions on Software Engineering, 28 (1). 109-111.
23. Younis, M.I., Zamli, K.Z. and Isa, N.A.M., IRPS–An Efficient Test Data Generation Strategy for Pairwise Testing. in Proc. of the 12th Intl. Conf. on Knowledge-Based and Intelligent Information & Engineering Systems (KES2008), (Zagreb, Croatia, 2008), Springer-Verlag, 493-500.
24. Yu-Wen, T. and Aldiwan, W.S., Automating Test Case Generation for the New Generation Mission Software System. in Proc. of the IEEE Aerospace Conference, (Big Sky, MT, USA, 2000), 431-437.