An Effective t-way Test Data Generation Strategy

Khandakar Rabbi(✉) and Quazi Mamun

School of Computing and Mathematics, Charles Sturt University, Bathurst, Australia
{krabbi,qmamun}@csu.edu.au

Abstract. Software testing is an integral part of software development life cycle which ensures the quality of the software. An exhaustive testing is not always possible because of combinatorial optimisation problem. Thus, in the software testing phase, generation of optimal number of test data accelerate the overall software testing process. We identified that the reduction of interactions among the input parameters significantly reduces the number of test data and generate an optimal test data set. This interaction is known as 't'-way interaction. Over the last decade, a large number of 't'-way test data generation strategies have been developed. However, generating optimum number of test data appears to be a NP-hard problem where the test data generation time becomes significantly higher. This paper proposes an effective test data generation strategy based on 'Kids Card' game known as MTTG. The proposed strategy significantly reduces the test data generation time. The result and discussion section shows that, MTTG outperforms all other strategies.

Keywords: t-way testing · Test data generation strategy · Test optimization · NP-Hard problem

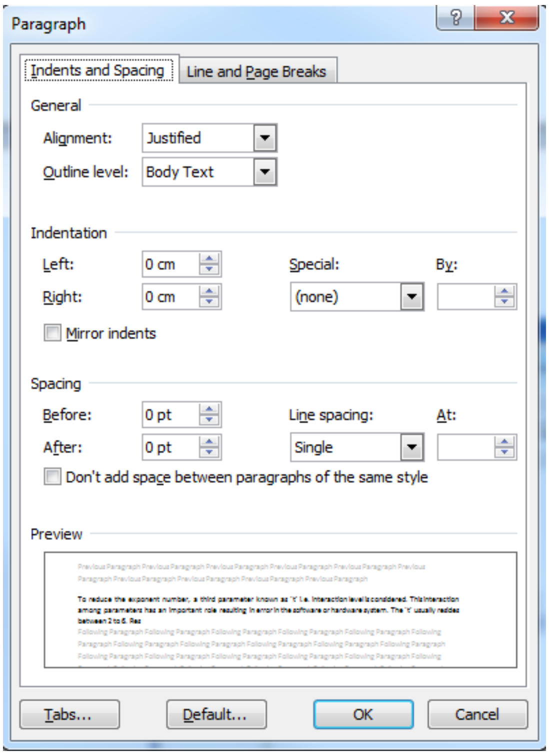
1 Introduction

On 4th June 1996, the European Space Agency launched the maiden flight of the Ariane 5. But it exploded 40 seconds after lift-off at an altitude of 3700 m. This accident was investigated by the Massachusetts Institute of Technology research team. Their report indicated that, a component was erroneously putting a 64-bit floating number into a 16-bit floating number. This eventually causes overflow error which affects rocket alignment [1]. This error was caused by lack of software testing which can be disastrous and life threatening.

About 50% of the total cost and resources are allocated to software testing which is considered an important and integral part of the software develop life cycle. Paying attention to the software testing can lead to an overall reduction in costs. The cost reduction can be achieved through process automation. However, an optimum and effective test data set by reducing the amount of test data required can also reduce the overall software testing costs [4-21]. To understand what the test data is and its magnitude, let's consider a very simple system having 5 parameters with 10 values each. It produces 10^5 number of test data. To a further extend, if we consider Figure 1, which is a single *'Indents and Spacing'* under the *'Paragraph'* dialog in '*Microsoft Word*'. It consists of non-uniform parameterized values i.e. one parameter 'Alignment' which has four values, '*Outline level*' has 10 values, '*Indentation Special*' has

© Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2015
B. Thuraisingham et al. (Eds.): SecureComm 2015, LNICST 164, pp. 633–648, 2015.
DOI: 10.1007/978-3-319-28865-9_42

two values and ‘*Line Spacing*’ has six values. Therefore, this single tab will have about 1^4 x 1^{10} x 1^2 x 1^6 = 480 numbers of test data. A manual testing will take about 24 hours to complete the testing of this tab [22]. When the system becomes more complex, number of test data increase exponentially.


Paragraph

Indents and Spacing | Line and Page Breaks

General
Alignment: Justified
Outline level: Body Text

Indentation
Left: 0 cm
Right: 0 cm
Special: (none)
By:
Mirror indents

Spacing
Before: 0 pt
After: 0 pt
Line spacing: Single
At:
Don't add space between paragraphs of the same style

Preview
Previous Paragraph Previous Paragraph Previous Paragraph Previous Paragraph Previous Paragraph Previous Paragraph Previous Paragraph Previous Paragraph Previous Paragraph Previous Paragraph
To reduce the exponent number, a third parameter known as 't' i.e. Interaction level is considered. This interaction among parameters has an important role resulting in error in the software or hardware system. The 't' usually resides between 2 to 6. Res
Following Paragraph Following Paragraph Following Paragraph Following Paragraph Following Paragraph Following Paragraph Following Paragraph Following Paragraph Following Paragraph Following Paragraph Following Paragraph Following Paragraph Following Paragraph Following Paragraph Following Paragraph Following

Tabs... Default... OK Cancel

Fig. 1. Paragraph dialog box in Microsoft word.

To reduce the exponent number, a third parameter known as ‘t’ i.e. interaction level is considered. This interaction among parameters has an important role resulting in error in the software or hardware system. The ‘t’ usually resides between 2 to 6. Research indicates that the appropriate reduction of the ‘t’ significantly reduces the number of test data by maintaining the standard quality. When the value of the ‘t’ is 2, it is known as 2-way testing or pairwise testing. On the other hand, when ‘t’ is greater than 2 ($t > 2$), it is known as t-way testing. The value of ‘t’ ranges from 2 up to a maximum number which is equal to the number of input variables. In the field of software testing, it is referred to as t-way testing.

Researchers have developed many t-way test data generation strategies to optimize the number of test data including OA [22], CA [23], MCA [24], TConfig [25], CTS [26], AllPairs [27], AETG [28], mAETG [29], TCG [30], mTCG [31], GA [14], ACA [14], IPO [32], IPOG [3], Jenny [20], TVG [19], ITCH [33], GTway [34], PSTG [35]. A brief description and scrutinizing analysis has been conducted throughout the appropriate section of this paper. Our empirical analysis identifies a basic problem in current test data generation strategies. Exhaustive analysis of test data produces combinatorial explosion problem (CEP) [3-21] which is also a NP-hard problem in common scientific and mathematical practice [5-21]. Thus, no abovementioned strategy can produce optimum number of test data in every input configuration. In addition to that, we also identifies that the complexity of the algorithm is very high and take non-polynomial time to generate the optimum test data set. Much effort has been expended to optimize the principal problem (CEP) through traditional computing analysis over the past decade [29-32]. However, through parallelization, CEP may be alleviated, but the development of complex software and hardware still poses the same question to the researchers. In addition, the parallel computing for test data generation is an expensive solution. Apart from this, the problem is also known as the NP-hard problem, where it is impossible to produce the optimum solution in every case (because of the nature of the problem itself). However, our study shows that most of the strategies take substantial time to produce the optimal test data. We have also identified the following research question [22, 29-37]:

1. What is the optimal and smaller set of test data to choose over the large dataset i.e. what strategy to choose that can produce optimal test data set?
2. Which test data generation strategy to choose in terms of complexity i.e. which strategy to choose that can produce faster test data?
3. What strategy to choose that supports maximum interaction level?

In the next section, we examined the available t-way test data generation strategies and explored the significance of generating a faster test data generation strategy.

2 Literature Review

Many attempts are taken to classify the existing t-way and pairwise test data generation strategies. *Cohen et. al.* has classified the number of test date generation strategies mainly into two groups (Cohen et. al. 2004). i) algebraic strategies ii) computational strategies. Grindal et. al. extends and expands the abovementioned strategies and identified three main sub-categories based on the randomness of the solution of the strategy: i) Non-deterministic ii) Deterministic iii) Compound. Non-deterministic strategies always produce random number of test data in each execution. It employs a random selection of test data over the search space. Artificial intelligence strategies are found to be non-deterministic. Thus, each solution produces different number of test data. On the other hand deterministic strategies appear to be producing same test data set in each execution. Usually, algebraic strategies found to be deterministic. Compound strategies are the combination of both deterministic strategy and non-deterministic strategy. The following sub-section analysied available 't' way test data generation strategies.

2.1 Analysis of Test Data Generation Strategies

There are few strategies which uses arithmetic operation to generate test. These strategies are usually arithmetic strategies. All most all of these strategies are limited to 2-way interaction level. To generate test data, these strategies are based on OA (Orthogonal Array), CA (Covering Array) and Mixed Level Covering Array (MCA). Orthogonal Arrays (OA) uses few different algebraic and the mathematical concepts [22]. This strategy uses 'Latin squares' to generate test data which significantly used in compiler design [22]. Analysis shows that OA strategy is deterministic. But the biggest impediment of orthogonal array is it's limitation to pairwise test data generation. It only uses symbolic data and no real data is uses as a part of data generation. Thus the practitioner's require mapping the real data with the symbolic data before operating the strategy. In addition, OA cannot support non-uniform input configuration which means each parameters require same number of values. Having the similarity with orthogonal array, CA is another form of array which can generate test data set. The major different of CA is it reduced the restriction of $\lambda=1$ which has been mentioned in other section. CA is also a deterministic approach. The major difference between CA and OA is CA supports 3-way test data generation where OA only supports pairwise or 2-way test data generation. Similar to OA, CA also cannot support non-uniform values. In addition, the strategy doesn't consider real input data as part of test data generation.

William et al. in 2000 proposed a computational tool using both OA and CA. He proposed an algorithm that can generate OA which in terms can be used as an initial block of larger CA. Thus his proposed algorithm uses both algebraic and combinatorial approach to generate test data set. TConfig is a deterministic approach. Although it uses the basics of OA and CA, it can support non-uniform values. It overcomes the limitation of CA and OA, however it is still limited to 6-way test data generation. Input configuration can be both symbolic and real data. Combinatorial Test Services (CTS) uses algebraic recursion as part of the generation of test data set. The algorithm uses C++ programming language. It is also referred as combinatorial recursive construction. It analyse all the possible input configurations. Based on the configurations, it selects the best covering array. The covering array can generate best test data set. CTS is a deterministic approach with the support of both uniform and non-uniform input configurations. However, input configurations can only be index values, thus no actual data can be used as a part of test data generation. Considering interaction level, CTS only supports only 2-way and 3-way. There are no published works found on AllPairs. It is mostly a tool developed in Perl (programming language) by Bach et al. in 2004. Later on, Cunningham developed a Java version of the tool. The tool only supports pairwise test data generation. The complexity of the tool is low. The tool generates test data in a deterministic approach. The tool supports both index values and real values as part of test data generation. In addition, the tool also supports non-uniform parameterized values. AETG starts with empty test data set and then add as many test data in the empty set. Finally choose the best test data which covers the most interaction levels. Our observation states that, AETG is the first computational strategy proposed by Cohen et al. in 1994. Later on few modifications alone with comparative results were shown in different publications. Analysis shows that AETG is a random approach which means it generates different number of test data set in different execution. Though the authors claim that AETG supports general t-way strategies but the publish results was limited to pairwise and 3-way. Input configuration was limited to index values, thus there were no supports for real data. However, AETG supports non-uniform values. A modified version of AETG was proposed by

Myra in her PhD thesis (2004). She has shown two basic difference of mAETG as compared to AETG. First difference was the randomness. Although AETG was non-deterministic, the number of test data was same for same configuration (the test data set was different). mAETG has a variable number of test data, which means it generate different number of test data for same input configuration in different execution. Second difference was the way to choose uncovered pairs. AETG was selecting the covered pairs first then later on it chooses the uncovered pairs randomly where mAETG chooses highly covered pairs first then it fix some variable to choose the uncovered pairs. Like AETG, mAETG is non-deterministic. mAETG can only support pairwise and 3-way test data generation. Input configurations are index values and no support for actual data. mAETG also supports non-uniform values.

TCG is a deterministic strategy. Yu-Wen et al. used TCG in 2000 used as a test data generator in 'Jet Propulsion Laboratory'. The algorithm used in TCG is similar to AETG which first generates empty test set and then add single test data until all t-way interaction is covered. Despite of that similarity, the test data generation is TCG always generates same test data in the same input configuration each time. TCG only supports pairwise test data generation. For the input configuration, it can only be symbolic. There is no support for actual data to be used. However, the strategy appears to be supporting non-uniform values. Similar to mAETG, Myra modified the original TCG and proposed mTCG. She modified the original rule based test data selection process to random based test data selection process. In the test data generation process, when mTCG finds the same test data covering similar number of pairs, mTCG choose any one randomly. Since mTCG uses random selection of test data, it is a non-deterministic approach. The strategy is limited to pairwise thus there is no support for 3-way. Input configuration can only be index values and there are no supports for actually data. However, mTCG can support non-uniform parameterized values.

For the first time, Shiba et al. in 2004 modified the original AETG and proposed an artificial intelligent based strategy known in test data generation. Each test data in GA is defined as chromosome. A number 'm' will generate randomly which is known as candidate test data. These test data will loop through an evaluation process. After that there will be crossover and mutation candidates based on few criteria and finally a set of test data will be chosen from that candidate set. Our analysis shows that GA is non-deterministic. Regarding t-way interaction, GA can only support t up to 3 levels. About the input configuration there is no support for actual data to be used as a part of test data generation. However, test data generation from non-uniform values are also supported by GA. Apart from GA, Shiba et al. worked on other artificial intelligent based strategy and implemented artificial ant colony algorithm in test data generation. ACA also used AETG as a base algorithm to generate test data. Implementation of ACA is a motivation of nature and an understanding how ants select their best path in orders to find out foods from various locations. ACA is a random search process thus appears to be non-deterministic. It can only supports pairwise and t = 3. There is no supports for real data to be used as a part of test data generation. However, ACA also supports non-uniform parameterized values.

IPO is a deterministic approach and the test data generation of IPO is very fast comparing to other test data generation strategies. It was first implemented in a tool called PairRest. IPO first generate an exhaustive number of test data from the first two pairs. After than other parameters are added by checking that if that parameter's value is paying the highest number of coverage or not. In the way it adds new values at the end of the each test data set and completes a test data set. IPO generates same number of test data in same input configuration which is a deterministic approach. IPO only support pairwise test data generation thus no supports for t=3. In input configuration IPO cannot support real input values and only supports index values. Our analysis also shows that IPO supports non-uniform values. IPOG is a basic strategy which is implemented in a popular tool known as FireEye. Development of FireEye tool was a collaboration work among ITL (Information Technology Laboratory) and NIST (National Institute of Standard and Technology) and the University of Texas, Arlington. The basic of IPOG is similar to IPO. It is developed to support higher t which was not supported in IPO. IPOG is a deterministic strategy and the supporting interaction level is up to 6. Input configuration can only be symbolic and no original data can be used as a part of test data generation. However, IPOG supports non-uniform parameterized values.

Jenkins in 2003 proposed a tool to generate test data which is known as Jenny. Jenkins stated that, Jenny starts generating test data by covering 1-way first, then 2-way, after that 3-way and until the proposed t-way. After generating 1-way, it checks if all 2-way has been covered or not. And when it covers 2-way, it checks that all 3-way has been covered or not. This is the way when the defined t-way covered, it release the test data set. Jenny produces same number of test data every time, shows that it is deterministic. Regarding interaction level, Jenny supports ‘t’ up to 8. About input configuration, Jenny doesn’t support original input as a part of test data generation. Jenny also supports non-uniform values. Test Vector Generation (TVG) is a tool proposed and developed by Schroeder et al. in 2003. The tool consists of three techniques. In the first technique, it produces test data randomly which supports only pairwise interactions. On the secondly technique, it was extended to support higher t-way interaction. And in the third technique, TVG uses an input and output relationships to reduce the number of exhaustive test data, hence generate the complete test data set. TVG is a deterministic strategy. As mentioned previously, TVG’s second technique supports higher ‘t’, however it is limited to ‘5’ level only. In the input configuration, both real input and symbolic input can be uses in the part of test data generation. In addition, TVG can also support non-uniform parameter values.

IBM developed test data generation strategy which is known as ITCH. And the windows version of the tool is known as WITCH. In ITCH, user can specify the number of test data. Based on that number ITCH, choose the proper interaction levels. Users can also specify the ‘t’ levels, which in terms can generate the number of test data set. It appears that ITCH is a deterministic strategy. Our observation also stated that, ITCH can support only 4 levels of interaction. Input configurations can be both symbolic and real data. In addition, ITCH can support non-uniform parameterized values. Klaib et al. in 2009 proposed a backtracking based test data generation strategy. It uses the basic IPO to choose the best coverable test data set. Once all the interactions are covered, it uses a backtracking algorithm to choose other test data set. He has also provided automation support in test data execution. Our analysis found that, GTway can support as much as 12 interaction levels. Input configuration can be both symbolic and real data. In addition, GTway also supports non-uniform parameterized values.

3 Design of MTTG

The proposed strategy has been created inspiring kids “Set” game, where the deck has total of 81 cards varying in four features: number (one, two, or three); symbol (diamond, squiggle, oval); shading (solid, striped, or open); and color (red, green, or purple) There are various versions of the game is available. However, in the most playing game, a player randomly take cards from the deck and try to make a complete ‘Set’ of a particular card by transferring to other members. The proposed strategy utilises the same strategy used in the ‘Set’ game. The cards are illustrated as the individual test data. Each test data is categorised by the combination of different parameters. A unique strategy runs over all the parameters, identifies the missing parameter and replaces it with a most effective parameter. The overall design can be divided into 3 major steps.

3.1 Step 1: Development of N-Tuples:

In first steps, the strategy reads the number of parameters and values and creates the N-Tuples. The number of N-Tuples depends on the ‘t’ i.e. interaction level. Figure 2 shows an N-Tuples generated from 3 parameters and 2 values in a 2-way/pairwise interaction.

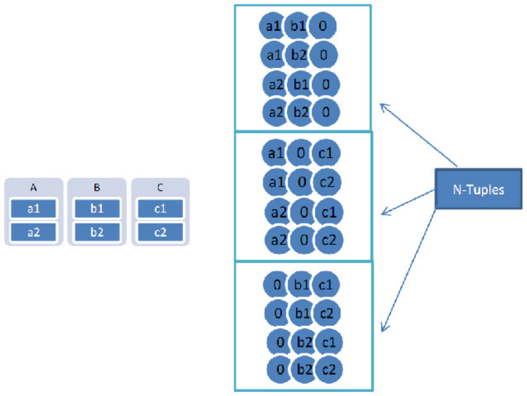


Fig. 2. Illustration of N-Tuples

The following equation has been used when creating N-Tuples:

$$N = \mathrm{p}!\frac{t!}{(P-t)!}$$

Where, N denotes N-Tuples, t denotes the interaction level and P is the number of parameters.

3.2 Step 2: Identification of ‘Missing Parameter’

In the second step, MTTG reads all the N-Tuple values. It identifies the missing parameter and adds 0 to the missing parameter of that Tuple. Thus MTTG an iteratively search the missing parameter and can replace it with the best possible value to produce the ‘Set’. Figure 3 illustrates the missing parameter.

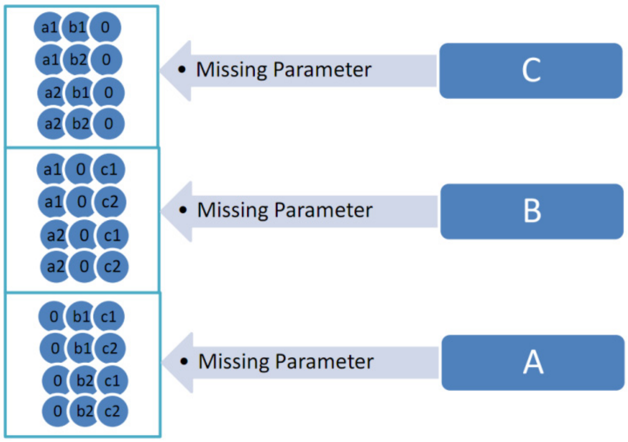


Fig. 3. Identification of Missing parameter

3.3 Step 3: Updating the Missing Parameter

This step involves searching for the 0th parameter and replacing it with an appropriate value so that a best coverage is possible. Replacing is possible based on two selection criteria i) Appropriate parameter ii) Appropriate value of the parameter. The algorithm ‘Test data construction’ has been shown as a pseudocode in the Figure 4.

```
  Begin
  Let $N_T$ = {} as a dataset represents the N-Tuples
  Let $N_{ST}$ = {} as empty dataset represents the subset of $N_T$
based on specific single Tuple.
  Let $N_{TS}$ = {} as final test data set
For each value ‘N’ in $N_T$
      $N_{ST}$ = N
      For each value V in $N_{ST}$
      If V == 0
            Read position of ‘V’ as P
            Find position of Parameter from P
      End If
      End For
      For each values in P
            Replace 0 with the values
            Create test data C
            Calculate coverage of C = $P_C$
      If $P_C$ == ‘Acceptable Number’
            Add C to $N_{ST}$
      End if
      End For
  End For
  End
```

Fig. 4. Test data generation pseudo code.

4 MTTG Flowchart

The flowchart of MTTG has been shown in the Figure 5. It starts with 'Generate Pair' section when the pairs are generated based on the interaction level. N-tuples are generated based on a formula from the generated pair. The N-Tuples are iterated bases on the coverage. It reads 0th parameter, replace with a possible value and calculate the coverage. If the coverage is acceptable, the test data is added to the final test data set.

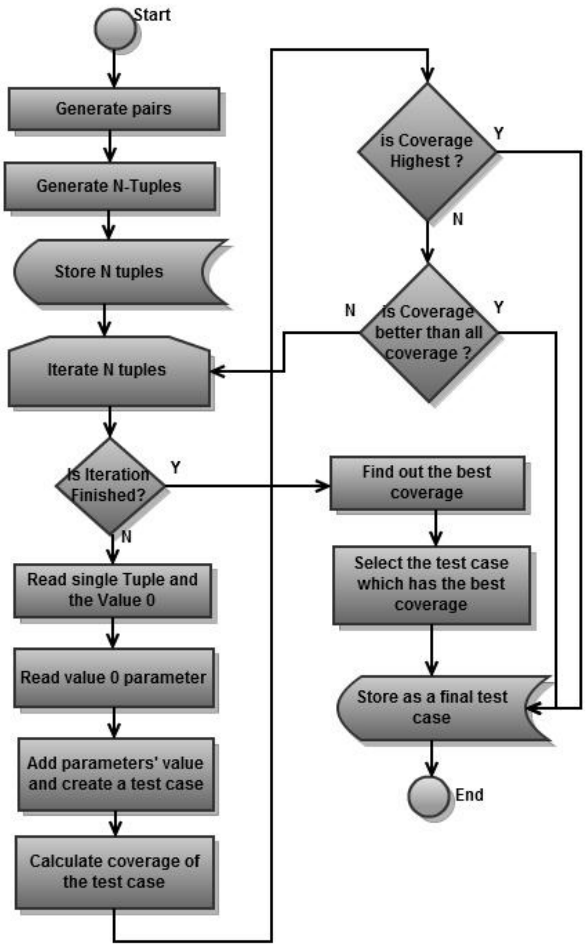

Start
Generate pairs
Generate N-Tuples
Store N tuples
Iterate N tuples
Is Iteration Finished?
Y
N
Read single Tuple and the Value 0
Read value 0 parameter
Add parameters' value and create a test case
Calculate coverage of the test case
is Coverage Highest ?
Y
N
Is Coverage better than all coverage ?
N
Y
Find out the best coverage
Select the test case which has the best coverage
Store as a final test case
End

Fig. 5. Complete workflow of MTTG

5 Results and Discussions

To evaluate the MTTG, we carried out a number of experiments both in terms of ‘Number’ of test data and the test data generation ‘Time’ i.e. complexity. The overall experiments are divided into four different groups:

G-1: ‘P’ and ‘V’ is constant, ‘t’ varies from 2 to 6.
G-2: ‘t’ and ‘V’ is constant, P varies from 5 to 15.
G-3: ‘P’ and ‘t’ is constant, V varies from 2 to 10.
G-4: TCAS dataset. 12 10-valued parameters, 1 4-valued parameters, 2 3-valued parameters and 7 2-valued parameters.

The results for test data size and complexity are separated into two tables for each group. Hence there are eight different tables have been used. The darken cell in each row represents the outperforming result. In some cases, there are more than one darken cell in each row means that more than one strategy have similar results. Cell marking NA (not available) indicates there are results unavailable or no published. NS (not supported) indicates that the strategy doesn’t support that specific configuration. Regarding complexity analysis, we were not able to run all the strategies into same platform however, a near proximity system configuration has been utilised for the evaluation.

Table 1a. Size for G-1
P & V constants (10, 5), but t varied up to 6

T-Way	IPOG	WHITCH	Jenny	Tconfig	TVG II	GTWay	MTTG
2	48	45	45	48	50	46	58
3	308	225	290	312	342	293	372
4	1843	1750	1719	1878	1971	1714	2194
5	10119	NS	9437	NA	NA	9487	11384
6	50920	NS	NS	NA	NA	44884	54166

Table 1b. Complexity (in Seconds) for G-1
P & V constants (10, 5), but t varied up to 6

T-Way	IPOG	WHITCH	Jenny	Tconfig	TVG II	GTWay	MTTG
2	0.11	1	0.43	1	0.141	0.265	0.019
3	0.56	23	0.78	88.62	5.797	6.312	0.193
4	6.38	350	17.53	>8hr	276.328	201.235	1.533
5	63.8	NS	500.93	>24hr	>24hr	3636.110	8.277
6	791.35	NS	NS	>24hr	>24hr	21525.063	24.719

Table 1a and 1b shows the result of G-1 in terms of Size and Time respectively. In terms of test data size WITCH, Jenny and GTway has outperform all other strategies. However, In terms of test data generation time, MTTG outperforms all others. The last row where ‘t’ = 6 shows a significant improvement of complexity comparing other strategies.

Table 2a. Size for G-2
t & V constants (4, 5), but P varied (from 5 up to 15)

P	IPOG	WHITCH	Jenny	Tconfig	TVG II	GTWay	MTTG
5	784	625	837	773	849	731	730
6	1064	625	1074	1092	1128	1027	1032
7	1290	1750	1248	1320	1384	1216	1321
8	1491	1750	1424	1532	1595	1443	1614
9	1677	1750	1578	1724	1795	1579	1890
10	1843	1750	1719	1878	1971	1714	2194
11	1990	1750	1839	2038	2122	1852	2485
12	2132	1750	1964	NA	2268	2022	2807
13	2254	NA	2072	NA	2398	2116	3165
14	2378	NA	2169	NA	NA	2222	3564
15	2497	NA	2277	NA	NA	2332	3884

Table 2b. Complexity (in Seconds) for G-2 t & V constants (4, 5), but P varied (from 5 up to 15)

P	IPOG	WHITCH	Jenny	Tconfig	TVG II	GTWay	MTTG
5	0.19	5.26	0.44	31.46	1.468	0.047	0.32
6	0.45	14.23	0.71	231.56	5.922	0.563	0.45
7	0.92	59.56	1.93	1,120	18.766	3.046	0.63
8	1.88	115.77	4.37	>1hr	55.172	15.344	0.88

Table 2b. *(continued)*

9	3.58	210.87	9.41	>3hr	132.766	63.516	1.28
10	6.38	350	17.53	>8hr	276.328	201.235	1.53
11	10.83	417	30.61	>23hr	548.703	599.203	2.94
12	17.52	628.94	50.22	>24hr	921.781	1682.844	4.71
13	27.3	>24hr	76.41	>24hr	1565.5	4573.687	7.40
14	41.71	>24hr	115.71	>24hr	>24hr	11818.281	11.96
15	61.26	>24hr	165.06	>24hr	>24hr	28793.360	18.74

Table 2a and 2b shows the result of G-2 in terms of Size and Time respectively. In terms of test data size, there is a uniformed distribution was found. Almost all strategies have achieved good results into a particular configuration. However, In terms of test data generation time, MTTG outperforms all others.

Table 3a. Size for G-3
P & t constants (10, 4), but V varied (from 2 up to 10)

V	IPOG	WHITCH	Jenny	Tconfig	TVG II	GTWay	MTTG
2	46	58	39	45	40	46	50
3	229	336	221	235	228	224	277
4	649	704	703	718	782	621	1950
5	1843	1750	1719	1878	1971	1714	2194
6	3808	NA	3519	NA	4159	3514	4531
7	7061	NA	6482	NA	7854	6459	8245
8	11993	NA	11021	NA	NA	10850	13928
9	19098	NA	17527	NA	NA	17272	21944
10	28985	NA	26624	NA	NA	26121	32966

Table 3b. Complexity (in Seconds) for G-3 (Time)
P & t constants (10, 4), but V varied (from 2 up to 10)

V	IPOG	WHITCH	Jenny	Tconfig	TVG II	GTWay	MTTG
2	0.16	1	0.47	14.43	0.297	1.282	0.04
3	0.547	120.22	0.51	379.38	3.937	7.078	0.18
4	1.8	180	4.41	>1hr	46.094	25.250	1.34
5	6.33	350	17.53	>8hr	276.328	201.235	1.69
6	16.44	>24hr	134.67	>24hr	1,273.469	765.453	3.81
7	38.61	>24hr	485.91	>24hr	4,724	2389.812	6.78
8	83.96	>24hr	1410.27	>24hr	>24hr	6270.735	10.66
9	168.37	>24hr	2125.8	>24hr	>24hr	15672.531	16.18
10	329.36	>24hr	5458	>24hr	>24hr	35071.672	24.28

Table 3a and 3b shows the result of G-3 in terms of Size and Time respectively. In terms of test data size GTway has outperform almost all other strategies. However, In terms of test data generation time, MTTG outperforms all others.

Table 4a. Size for G-4
TCAS Module (12 multi-valued parameters, t varied from 2 to12)

T-Way	IPOG	WHITCH	Jenny	Tconfig	TVG II	GTWay	MTTG
2	100	120	108	108	101	100	100
3	400	2388	413	472	434	402	406
4	1361	1484	1536	1476	1599	1429	1404
5	4219	NS	4580	NA	4773	4286	4355
6	10919	NS	11625	NA	NS	11727	13667
7	NS	NS	27630	NS	NS	27119	35313
8	NS	NS	58865	NS	NS	58584	70600
9	NS	NS	NA	NS	NS	114411	127811
10	NS	NS	NA	NS	NS	201728	212400
11	NS	NS	NA	NS	NS	230400	230400
12	NS	NS	NA	NS	NS	460800	460800

Table 4b. Complexity (in Seconds) for G-4
TCAS Module (12 multi-valued parameters, t varied from 2 to 12)

T-Way	IPOG	WHITCH	Jenny	Tconfig	TVG II	GTWay	MTTG
2	0.8	0.73	0.001	>1hr	0.078	0.297	0.07
3	0.36	1,020	0.71	>12hr	2.625	1.828	0.13
4	3.05	5,400	3.54	>21hr	104.093	58.219	1.00
5	18.41	NS	43.54	>24hr	1,975.172	270.531	5.47
6	65.03	NS	470	>24hr	NS	1476.672	19.36
7	NS	NS	2461.1	NS	NS	4571.797	41.90
8	NS	NS	11879.2	NS	NS	10713.469	53.59
9	NS	NS	>1day	NS	NS	14856.109	45.29
10	NS	NS	>1day	NS	NS	10620.953	27.43
11	NS	NS	>1day	NS	NS	363.078	12.92
12	NS	NS	>1day	NS	NS	12.703	8.06

Table 4a and 4b shows the result of G-4 in terms of Size and Time respectively. In terms of test data size GTway and IPOG has better results than others. However, In terms of test data generation time, MTTG outperforms all others. Based on the results found in the above tables, an interesting observation can be summarized. It is clear that no single strategy has domination over others in terms of test data size. However, concerning test data generation time, MTTG is dominating in all the cases. On the other hand, WHITCH and TConfig appear to be a caterer for smaller configuration where ‘t’ is below 4. In addition to that, MTTG and GTway appear to be more effective for complex configurations. In terms of test data generation time, Table 3b shows the effectiveness of MTTG. In that scenario, GTway takes about 20 hours where MTTG takes less than 1 minute. Thus, concerning complex configuration MTTG is highly acceptable than all other strategies.

6 Conclusion

We propose MTTG (Multi-Tuple Test Generator) which is an effective test data generation strategy. The performance of the MTTG has been compared with other strategies in terms of test data size and time complexity. It is to remember that, the NP-hard problem prevented any strategy from outperforming others in terms of both efficiency and complexity. Thus our approaches involves in generating test data in most of the cases so that it can be acceptable in all aspect. In some cases, the testing professional often knows the importance of a particular parameter over others. Thus, it might be important to implement different interactions among different parameters. As an example, if A, B, C are three parameters containing 3 values each in a configuration, and parameter C is less important to consider then, a 3-way interaction might be require to apply between A and B where, A and C or B and C might require only a 2-way interaction.

References

1. Lions, J.L.: Ariane 5 Failure: Full Report. http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html (accessed February 11, 2015)
2. Patrick, M., Alexander, R., Oriol, M., Clark, J.A.: Subdomain-based test data generation. The Journal of Systems and Software, 1–15, November 2014. Elsevier
3. Lei, Y., Kacker, R., Kuhn, D.R., Okun, V., Lawrence, J.: IPOG: a general strategy for t-way software testing. In: 14th Annual IEEE International Conference and Workshops on the Engineering and Computer-Based Systems (2007)
4. Cui, Y., Li, L., Yao, S.: A new strategy for pairwise test case generation. In: 3rd International Symposium on Intelligent Information Technology Application (2009)
5. Chen, X., Gu, Q., Qi, J., Chen, D.: Applying particle swarm optimization to pairwise testing. In: 34th Annual IEEE Computer Software And Application Conference (2010)
6. Younis, M.I., Zamli, K.Z., Isa, N.A.M.: Algebraic strategy to generate pairwise test set for prime number parameters and variables. In: IEEE International Conference on Computer and Information Technology (2008)

7. Younis, M.I., Zamli, K.Z., Mat Isa, N.A.: IRPS – an efficient test data generation strategy for pairwise testing. In: Lovrek, I., Howlett, R.J., Jain, L.C. (eds.) KES 2008, Part I. LNCS (LNAI), vol. 5177, pp. 493–500. Springer, Heidelberg (2008)
8. Klaib, M.F.J., Muthuraman, S., Ahmad, N., Sidek, R.: A tree based strategy for test data generation and cost calculation for uniform and non-uniform parametric values. In: 10th IEEE International Conference on Computer and Information Technology (2010)
9. Shitao, W., Hao, W.: A novel algorithm for multipath test data generation. In: 4th International Conference on Digital Manufacturing & Automation (2013)
10. Cohen, D.M., Dalal, S.R., Kajla, A., Patton, G.C.: The automatic efficient test generator (AETG) system. In: 5th International Symposium on Software Reliability Engineering (1994)
11. Bach, J.: Allpairs Test Case Generation Tool. http://tejasconsulting.com/open-testware/feature/allpairs.html (access September 27, 2009)
12. Cohen, D.M., Dalal, S.R., Fredman, M.L., Patton, G.C.: The AETG System: An Approach to Testing Based on Combinatorial design. IEEE Transactions on Software Engineering (1997)
13. Lei, Y., Tai, K.C.: In-parameter-order: a test generation strategy for pairwise testing. In: 3rd IEEE Intl. High- Assurance Systems Engineering Symposium (1998)
14. Shiba, T., Tsuchiya, T., Kikuno, T.: Using artificial life techniques to generate test cases for combinatorial testing. In: 28th Annual International Computer Software and Applications Conference (2004)
15. Harman, M., Jones, B.F.: Search based software engineering. Information and Software Technology (2001)
16. Klaib, M.F.J., Zamli, K.Z., Isa, N.A.M., Younis, M.I., Abdullah, R.: G2Way – a backtracking strategy for pairwise test data generation. In: 15th IEEE Asia-Pacific Software Engineering Conference (2008)
17. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: IEEE international Conference on Neural Networks (1995)
18. TConfig: http://www.site.uottawa.ca/~awilliam/ (access September 27, 2010)
19. TVG: http://sourceforge.net/projects/tvg (access September 27 2010)
20. Jenny: http://www.burtleburtle.net/bob/math/ (access September 27, 2010)
21. Yan, J., Zhang, J.: A Backtracking Search Tool for Constructing Combinatorial Test Suites. Journal of Systems and Software – Elsevier (2008)
22. Chateauneuf, M., Kreher, D.: On the State of Strength-Three Covering Arrays. Journal of Combinatorial Designs (2002)
23. Colbourn, C.J., Martirosyan, S.S., Mullen, G.L., Shasha, D., Sherwood, G.B., Yucas, J.L.: Products of Mixed Covering Arrays of Strength Two. Journal of Combinatorial Designs (2005)
24. Williams, A.W.: Determination of test configurations for pair-wise interaction coverage. In: Proc. of the 13th International Conference on Testing of Communicating Systems (2000)
25. Hartman, A., Raskin, L.: Combinatorial Test Services (2004). https://www.research.ibm.com/haifa/projects/verification/mdt/papers/CTSUserDocumentation.pdf (accessed March 2015)
26. Bach, J.: ALLPAIRS Test Generation Tool, Version 1.2.1 (2004). http://www.satisfice.com/tools.shtml (accessed March 2015)
27. Ellims, M., Ince, D., Petre, M.: AETG vs. Man: an Assessment of the Effectiveness of Combinatorial Test Data Generation. UK, in Technical Report, Department of Computing, Faculty of Mathematics and Computing, Open University (2008)

28. Cohen, M.B., Dwyer, M.B., Shi, J.: Exploiting constraint solving history to construct interaction test suites. In: Proc. of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. IEEE Computer Society, UK (2007)
29. Yu-Wen, T., Aldiwan, W.S.: Automating test case generation for the new generation mission software system. In: Proc. of the IEEE Aerospace Conference (2000)
30. Cohen, M.B.: Designing Test Suites for Software Interaction Testing. Computer Science. New Zealand, University of Auckland (2004)
31. Forbes, M., Lawrence, J., Lei, Y., Kacker, R.N., Kuhn, D.R.: Refining the In-Parameter-Order Strategy for Constructing Covering Arrays. NIST Journal of Research (2008)
32. Hartman, A., Raskin, L.: Problems and Algorithms for Covering Arrays. Discrete Mathematics-Elsevier (2004)
33. Zamli, K.Z., Klaib, M.F.J., Younis, M.I., Isa, N.A.M., Abdullah, R.: Design and implementation of a t-way test data generation strategy with automated execution tool support. Information Sciences, Elsevier (2009)
34. Ahmed, B.S., Zamli, K.Z.: PSTG: A T-Way Strategy Adopting Particle Swarm Optimization. Mathematical/Analytical Modelling and Computer Simulation (AMS) (2010)
35. Khatun, S., Rabbi, K.F., Yaakub, C.Y., Klaib, M.F.J., Masroor Ahmed, M.: PS2Way: an efficient pairwise search approach for test data generation. In: Zain, J.M., Wan Mohd, WMb, El-Qawasmeh, E. (eds.) ICSECS 2011, Part III. CCIS, vol. 181, pp. 99–108. Springer, Heidelberg (2011)
36. Rabbi, K.F., Khatun, S., Yaakub, C.Y., Klaib, M.F.J.: EasyA: easy and effective way to generate pairwise test data. In: 2011 Third International Conference on Computational Intelligence, Communication Systems and Networks (2011)