MTTG: An efficient technique for test data generation

Conference Paper · December 2014

DOI: 10.1109/SKIMA.2014.7083520

4 authors, including:

Md Rafiqul Islam
Charles Sturt University
105 PUBLICATIONS **585** CITATIONS
SEE PROFILE

Quazi Mamun
Charles Sturt University
62 PUBLICATIONS **179** CITATIONS
SEE PROFILE

MTTG: An Efficient Technique for Test Data Generation

Khandakar Rabbi[1], Rafiqul Islam[1]
[1]School of Computing and Mathematics
Charles Sturt University
{krabbi,mislam,qmamun}@csu.edu.au

Quazi Mamun[1] and Mohammed Golam Kaosar[2]
[2] Department of Computer Science
Effat University, Jeddah, Saudi Arabia
mkaosar@effatuniversity.edu.sa

***Abstract*—Test data generation is a significant part of software and/or hardware testing. It is a complex problem and researchers have proposed various solutions to generate optimum number of test data in an acceptable polynomial time. However, most of the solutions are highly complex (NP-hard), interaction limitation and takes substantial time to generate test data set. Therefore, it is an open challenge to propose the best solution. This paper proposes a novel technique called MTTG (Multi-Tuple based T-way Test Generator) which relies on Kid's Card game strategy. The proposed technique finds optimum matching value by searching through all possible combination of similarity matching, based on data generation principle. Our empirical results demonstrate that the proposed MTTG is very efficient in test data generation based on time and interaction strength/level, compared to other existing strategies.**

Keywords—Combinatorial interaction testing; Software testing; T-way testing; Test case generation; Complete Test Data Generation

I. INTRODUCTION

In 1996, European Space Agency launched Ariane 5. Unfortunately it exploded only 37 seconds after liftoff [19]. Massachusetts Institute of Technology research team investigated the whole case and found the root cause. A component erroneously put 64 bit floating number into a 16 bit floating number which eventually caused overflow error, and therefore, it affected rocket alignment [19]. This incident shows that even a very small error of software testing can bring about a disaster and life threatening accidents.

Survey shows that about half of the total cost and resources are allocated for software testing and it is obviously an integral part of the software development life cycle [1, 18]. With a lot other testing methods, combinatorial testing is one of the good strategies to generate test data by selecting parameters and values [2]. For example, a system having 6 parameters and 10 values generates 106 exhaustive test data which is nearly impossible to execute because of time constraints, resource constraints, and time to market factor. Another example is shown here with Microsoft Excel. Consider the proofing tab under option dialog in Microsoft Excel (see Figure 1), it consists 6 possible configurations which needed to be tested. Two values for each checkboxes (checked or unchecked) beside there have the French modes which takes 3 possible values and finally Dictionary language takes 54 possible values. Thus only this proofing tab requires 26 × 54× 3 i.e. 10,368 test data to be executed. Assuming a manual testing may consume 4 minutes to execute each single test data; results around 28 days to complete the test of this single proofing tab [3].

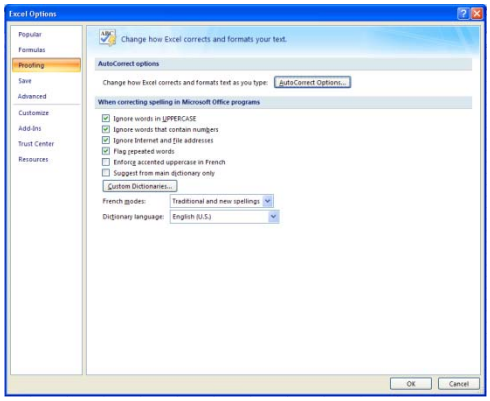

Figure 1: Microsoft Excel Proofing Option [21]

This is similar for hardware products. If a product has 30 on/off switches, to test all possible combinations requires 230 = 1,073,741,824 test cases. This may consume 170 years by considering 5 seconds for each case of manual testing [3].

Above examples are known as 'Combinatorial Explosion Problem' in software testing. Having the limited time and resource the common research questions are [21,22]:

1. What is the optimal and smaller set to test data?
2. What strategy can generate fast test data?
3. What interaction level should be chosen for testing?

In modern computing combinatorial explosion problem [21, 22] is known as one of the greatest challenges to overcome. Various strategies have been proposed to overcome this problem. Although parallel processing can overcome some of the time limitation, the complexity of the modern system is increasing as well [22]. International Software Testing and Quality Board (ISTQB) states that 'Exhaustive testing is impossible' [21]. Thus, there should be a strategy to produce quality test data in an acceptable polynomial time.

Authors in [11] state that the interaction among different input parameters causes the system failure. This interaction usually known as T where if 'T'=2 is known as pairwise testing. Having higher values of 'T' is usually known as 'T-way testing'.

978-1-4799-6399-7/14/$31.00 ©2014 Crown

A number of useful T-way strategies have been developed in the last decade including TConfig [29], CTS [30], AETG [31], mAETG [32], GA [33], ACA [33], IPOG [34], Jenny [35], TVG [36], GTWay [22] and PSTG [37]. Where some of these strategies use real data, most of them use symbolic/syntax data to produce test data. In real testing phase, the test data converted into real system data to execute against the system. Among all of the available strategies, GTWay [22] is the only strategy which supports highest interaction but limited to 12. Researchers found that the production of minimum number of test data is NP-complete [29-36]. In general, any specific strategy cannot always produce minimum number of test data in a minimum time.

Keeping above problems and research questions in mind, we investigate all the above techniques and propose an efficient solution called MTTG (Multi-Tuple based T-way Test Generator). Our proposed technique relies on Kid's Card game strategy [20], which is a rule to match the similarity among the cards, based on color and shape, and pick the best matching one. Our technique MTTG adds a setup value to all the initial matching combination and searches for the optimum matching. Once the technique reaches the optimum matching, based on data searching principle, it replaces the initial value with the optimum value and generates test data set. Our empirical results demonstrate that that the proposed MTTG is more efficient in test data generation based on time and interaction strength/level, compared to existing strategies.

The rest of the paper is organized as follows: the next section investigates all available T-way strategies. In section 3, we describe the proposed strategy. Section 4 discusses the performance of MTTG and finally a conclusion and further works ends the paper.

II. Related Work

Test Data generation strategies are divided into three major categories: i) Algebraic:- by implementing mathematical equations ii) Computational:- by implementing computer processing iii) Artificial Intelligent:- by implementing artificial intelligent like neural networks, genetic algorithms or particle swarm optimization algorithms[5, 6, 21].

A. Algebraic Strategies

Algebraic strategies [31, 32] are capable of generating optimum number of test data which have some redefined conditions. These strategies uplift the mathematical equations to some extent. Usually they are faster than all other strategies because those do not require using the computational power. Despite of having this advantage, these strategies are only limited to pairwise interaction level. Test Configuration (TConfig) is one known tool available which implements mathematical equations [29].

B. Computational Strategies

Computational strategies usually take longer time than algebraic strategies to generate test data but it does not yield the optimum number of test data. But almost all the available computational strategies support higher T. In Parameter Order (IPO) strategy supports pairwise testing [38]. Input values are presented by the index values (0, 1, 2 ....) which are known as symbolic inputs. Our analysis shows that IPO is faster than most of the other computational strategies. IPO is also deterministic meaning that, the production of test data is same every time. AllPairs strategy only supports pairwise testing [39]. This strategy is also deterministic. The input values are represented through index (0, 1, 2 ....). In addition, AllPairs supports actual values for test data generation. This strategy supports a different number of values in input parameters (non-uniform). Test Case Generator (TCG) is a deterministic strategy [40] where input values are presented as index (0, 1, 2 ... etc.). It does not support actual data as a part of test data generation. In addition, TCG also supports different number of values in different number of parameter (non-uniform). modified Test Case Generator (mTCG) strategy is a non-deterministic approach [32] where input values are presented through index (0, 1, 2, ...) without actual data support. In addition, it supports non-uniform values. Combinatorial Test Services (CTS) is a deterministic strategy [30]. In this strategy, a covering array concept in used to generate series of test data. It supports 'T' at 3 levels. Input values are presented through indexes (0, 1, 2 ....). It does not support actual data as a part of test data generation. It supports non-uniform inputs. Automatic Efficient Test Generator (AETG) strategy uses random approach making it non-deterministic [31]. Input values are presented through index (0, 1, 2 ....). Alike with others, it does not support actual input data as part of test data generation. However, it can support non-uniform input values. modified Automatic Efficient Test Generator is a strategy which is similar to AETG. This strategy uses a random generation for test data [31, 32]. Thus, the test data is non-deterministic. However, this strategy is only limited to support T = 2 and 3. Input values are presented through index (0, 1, 2, ...), without actual data used as input during test data generation. This strategy appears to support non-uniform parameters. Generation of In Parameter Order (IPOG) strategy is deterministic [34] which supports higher 'T' where T = 6. The input values are limited to use indexes (0, 1, 2 .....). In addition, it supports non-uniform values. Jenny is a deterministic test data generation strategy [35]. Jenny supports much higher T compare to other strategies. It supports 'T' up to 8 levels. Input values are presented through symbolic index. Jenny supports non-uniform values. Test Vector Generator (TVG) is a deterministic strategy [36]. But it cannot support T more than 5. But it uses both actual data and symbolic index to generate test data. In addition, it supports non-uniform values. Intelligent Test Case Handler (ITCH) is a deterministic strategy [30]. ITCH supports T up to 4 levels. It supports both symbolic index and actual inputs as a part of test data generation. It supports non-uniform input values. Generation of Test Case (GTWay) is a deterministic strategy [22]. This strategy supports T up to 12 levels. Both real inputs and indexes are used as a part of test data generation. GTWay also supports non-uniform input values.

C. Aritificial Intelligence Strategies

Alike with computational strategy, artificial intelligence strategies appear to be non-deterministic which may produce optimum test data in some cases. Shiba in 2004 first introduce artificial intelligence through implementing Genetic Algorithm

(GA) [33] and Ant Colony Algorithm (ACA) [33]. In 2010, Bestoun S. uses Particle Swarm Intelligence (PSO) to generate test data in [37]. In GA (Genetic Algorithm), each chromosome treats as a test data [33]. Few numbers of chromosomes are generated in a random fashion. The generated chromosomes are also known as candidate test data. These candidate test data are passes through an evaluation process to check its fitness. If fitness is beyond a threshold, it is then selected as a final test data. However, it only supports T = 3. It is only limited to symbolic input values and no support for actual data as a part of test data generation. It supports non-uniform input. Ant Colony based Algorithm (ACA) is a natural optimization. In ant colony algorithm, every path is known to be a test data. All paths are evaluated and the best path is selected as a part of test data. ACA does not appear to support higher T values. It can support non-uniform values. Particle Swarm Optimization (PSO) has been used for the last few years as a popular optimization method. It uses natural optimization for searching in a large space. PSTG is a strategy which uses PSO as a test data generation strategy [37]. It supports T up to 6 levels. It uses symbolic values as part of test data generation but no support for actual data. It also supports non-uniform values.

D. Comparison Of All Strategies

An illustration and comparison of all the strategies are shown in Table 1. The comparison carries different T levels supporting by a particular strategy and whether they support real value or index value.

TABLE 1. COMPARISON OF DIFFERENT STRATEGIES.

Strategy	T-Way support	Real value support
IPO [38]	2	No
AllPairs [39]	2	No
TCG [40]	2	No
mTCG [32]	2	No
TConfig [29]	6	No
CTS [30]	3	No
AETG [31]	3	No
mAETG [31]	3	No
IPOG [34]	6	No
Jenny [35]	8	No
TVG [36]	5	Yes
ITCH [30]	4	Yes
GTWay [22]	12	Yes
GA [33]	3	No
ACA [33]	3	No
PSO [37]	6	No

From Table 1, it can be state that most of the strategies do not support higher interaction (T). GTway is the one which can only support T = 12. A system consist of 100 inputs and may require a much higher 'T' which can be as much as T= 40. Any of the abovementioned strategy will not be able to produce test data for such a system.

Motivated from above problem, we inspire to create a new strategy which is less complex and can support higher T. In addition, considering the test data a generation time, most of the strategies are highly complex and takes unacceptable amount of time. To overcome these problems we propose a new strategy which describes in the next section.

III. PROPOSED MTTG ALGORITHM

To demonstrate how MTTG works, we divided our strategy in 4 different algorithms. 1. Generation of 'SET Hand', 2. Generation of 'Missing Card', 3. Addition of best possible 'Missing Card' 4. Adjustment of 'Uncovered Combination'.

A. Generation of 'SET Hand'

This part illustrates the cards shuffling. To understand the fact, let's consider the following example:

- Parameter A has 3 values a1, a2, and a3
- Parameter B has 2 values b1, b2
- Parameter C has 4 values c1, c2, c3, and c4

This can be an illustration of 'SET GAME' having following characteristics:

- A is a SET having 3 values a1, a2, a3
- B is a SET having 2 values b1, b2
- C is a SET having 4 values c1, c2, c3, c4

It is understandable that 2-way parameter combination can be 3, which can be AB, AC and BC. This can be illustrating as the number of player in our proposed SET strategy. This part of the strategy stores combination of parameters and combination of corresponding values as shown in Table 2.

TABLE 2. GENERATED OF SET HAND

Combination of Parameters	Combination of Corresponding Values
AB	(a1, b1), (a1, b2), (a2, b1), (a2, b2), (a3, b1), (a3, b2)
AC	(a1, c1), (a1, c2), (a1, c3), (a1, c4), (a2, c1), (a2, c2), (a2, c3), (a2, c4), (a3, c1), (a3, c2), (a3, c3), (a3, c4)
BC	(b1, c1), (b1, c2), (b1, c3), (b1, c4), (b2, c1), (b2, c2), (b2, c3), (b2, c4)

The generation of combination can be express as:

$Cv = [Nv1 * Nv2]$

Where,

Cv= Combination of Values,

Nv1 = Number of values in first parameter,

Nv2 = Number of values in second parameter.

Once when the combinations are ready, it is ready for second level processing. Next section describes the further processing of this combination which is known as 'Generation of Missing Card'.

B. Generation of 'Missing Card'

This part of the strategy adds '0's as a missing part. The understanding behind it is that, later on a perfect value should be replaced to complete the test data. One point need to consider how many '0' should one add? It is to understand that a complete test data requires a value from A, a value from B and a value from C (in this example). Thus we have to add as many '0' as it completes a SET (ABC, in this case). Consider the first value from Table 2 which is (a1, b1). Thus to complete the SET it requires to add a '0' to complete a Test Data which is (a1, b1, 0) which in term means that some value from parameter C should come and join to make a complete test data. Table 3 illustrates the missing value combination.

TABLE 3. GENERATED MISSING VALUE SET FROM TABLE II

Combination of Parameters	Combination of Corresponding Values with missing value
AB	(a1, b1, 0)
	(a1, b2, 0)
	(a2, b1, 0)
	(a2, b2, 0)
	(a3, b1, 0)
	(a3, b2, 0)
AC	(a1, 0, c1)
	(a1, 0, c2)
	(a1, 0, c3)
	(a1, 0, c4)
	(a2, 0, c1)
	(a2, 0, c2)
	(a2, 0, c3)
	(a2, 0, c4)
	(a3, 0, c1)
	(a3, 0, c2)
	(a3, 0, c3)
	(a3, 0, c4)
BC	(0, b1, c1)
	(0, b1, c2)
	(0, b1, c3)
	(0, b1, c4)
	(0, b2, c1)
	(0, b2, c2)
	(0, b2, c3)
	(0, b2, c4)

Once the '0's are added, it is ready for next section of work. In that last section, the strategy evaluate all possible values replace the best value of corresponding parameter and construct a complete test data.

C. Addition of best possible 'Missing Card'

In this part of the strategy, 'Missing Cards' are replaced with all possible values. This section starts reading the first combination which is (a1, b1, 0) in this example. It tries to replace all the value of parameter C which can be c1, c2, c3 or c4. Once it finds the best value it replaces '0' with the best value. It is to note that, the best value depends on the best coverage. The following table shows the pseudo-code for replacing 'Missing Card'.

TABLE 4. ALGORITHM FOR REPLACING MISSING CARD

```
Begin
   Let AVL = {} as set, where AVL represents the full
missing value set
   Let CRL = {} as empty set, reads current record
set
   Let CTS = {} as empty set, stores a Test Data
   Let PC is a integer represents number of pair
coverage
For each value v1 in AVL records
   CRT = v1
   For each value v2 in CRT
       If v2 is equals to '0'
           read v2 position as P
          Find parameter position from P
              For each values in P
                replace 0 with the values
                create test case set as CTS
                Get pair coverage of CTS = PC
                  If PC is the highest pair Coverage
                    select CTS as the final test data
                  Else
                    Get current coverage = PC.
                  End If
               select CTS as the final test data
         End if
    End For
End For
End
```

Through replacing '0's the strategy can construct initial test data set. The next section of the strategy keeps a vital role which makes sure that all coverage is complete. Most of the time, missing card strategy can't cover all the combinations. Next section describes the last part of the strategy.

D. Adjustment of 'Uncovered Combination'

The purpose of adjustment algorithm is to combine the uncovered combinations and try to merge each other to create another test data. It iteratively reads all other combinations. Table 5 shows a pseudo-code for adjustment algorithm.

TABLE 5. ALGORITHM OF ADJUSTMENT

```
Begin
   Let UC = {} as set, where UC represents the
Uncovered Combinations
   Let URL = {} as empty set, reads current record
set
   Let CTS = {} as empty set, stores a Test Data
   Let PC is a integer represents number of pair
coverage
For each value v1 in UC records
   URT = v1
   For each value v2 in UC + 1
```

```
        If v1, v2 adjustable?
           U_RL = adjust(v1, v2)
        End if
     End For
End For
End
```

Table 5 shows an iteration which reads all the uncovered combination and adjust each with others. The following rules applied for adjustment:

a) A, B parameter's values can adjust with any C values. If any uncovered combination is (a1, b1) then c1, c2, c3 or c4 can adjust with (a1, b1)

b) A, C parameter's values can adjust with any B values. If any uncovered combination is (a1, c1) then b1 or b2 can be able to adjust with (a1, c1)

c) B, C parameter's values can adjust with any A values. If any uncovered combination is (b1,c1) then a1, a2 or a3 can be able to adjust with (b1, c1)

IV. RESULT AND DISCUSSION

Since MTTG supports pairwise testing as well as higher T-Way. Thus, results and the discussions are shown in three subsections. In the next sub-section MTTG results have been compared with available 2-Way strategies and in the corresponding sub-section MTTG is compared with available T-Way strategies including IPOG [34], ITCH [30], Jenny [35], TConfig [29], TVG [36] and GTWay [22]. In the final subsection, a discussion shows how MTTG support higher T-way testing.

A. Experimental Setup

For the reasonable comparison, among the available strategies, we use the same platform used by Rabbi K. in [21]. The platforms for different strategies are as follows:

- JENNY, Tconfig, TVG: Intel P IV 1.8 GHz, C++ programming language, Linux Operating System.
- IPOG: Intel P II 450 MHz, Java programming language, Windows 98 operating system.
- WHITCH: Intel P IV 2.26 GHz, C programming language, Windows XP operating system.
- GTWay, MTTG: Intel P IV 1.6 GHz, 1 GB RAM, C++ programming language, Windows XP operating system.

B. MTTG vs Pairwise Strategies

To analysis the test data size, six system configurations evaluate, among those the first 3 are non-uniform parameterized values and the rest are uniform as follows:

- S1: 3 parameters with 3, 2 and 3 values
- S2: 3 parameters with 2, 1 and 3 values
- S3: 5 parameters with 3, 2, 1, 2 and 2 values.
- S4: 3 2-valued parameters
- S5: 3 3-valued parameters
- S6: 4 3-valued parameters.

The consideration of the parameters and assumptions are according to the following existing strategies that support pairwise testing: AETG [7, 8], IPO [10], AllPairs [14], Jenny [16], TVG [17] and ALL Pairs [14]. This is done to compare our results with those tools. Table 6 shows the generated test size by each algorithm. The darkened cell shows the best performance (in terms of test size). 'N' is data which is not capable of producing the test data.

Referring to the Table 6 (next page), it shows the proposed MTTG produces the comparable results in S1, S2, S3, S4 except S5 and S6. However test case production is a NP-complete problem and it is well known that no strategy may perform the best for all cases. Also TConfig shows the best result for all cases except S3. Hence MTTG is comparable with TConfig and outperforms others. Moreover, eight system configurations have been tested with the proposed strategy to compare the complexity. The following systems are:

- 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,
- S8: 1 5-valued parameters, 8 3-valued parameters and 2 2-valued parameters.

Table 7 (next page) shows the execution time in seconds for each system. The darkened cell entries row-by-row indicates the best performance (i.e. in term execution time). N is data that are not available to produce the result.

Concerning execution time, MTTG gives the best result in all systems. As oppose to S6 where every strategy takes too much long time except ALL Pairs. In all the system, MTTG generates less than 1 sec. Thus compare to execution time MTTG is much more acceptable than other strategies.

C. MTTG vs T-Way Strategies

To measure the efficiency of MTTG, a more complex system is which is widely known as 'The common Traffic Collision Avoidance System Module (TCAS)' with which consist of 12 parameters. (2 parameters with 10-values each, 1 parameter with 4-values each, 2 parameters with 3-values each, and 7 parameters with 2-values each) and interaction strength T varies from 2 to 12.

In the next page, Table 8 and Table 9 show the evaluation of MTTG with other T-way strategies. Only GTWay can support higher T level which is 12 in this case. Jenny is one which can support up to 6. However, GTWay takes long time to produce test data compare to MTTG. In interaction level 9 and 10 GTWay takes couple of hours where MTTG takes less than a minute.

TABLE 6. COMPARISON BASED ON THE TEST SIZE

Sys	AETG[7]	IPO[10]	TConfig[15]	Jenny[16]	TVG[17]	ALL Pairs[14]	MTTG
S1	N	9	9	9	9	9	9
S2	N	6	6	6	6	6	6
S3	N	7	8	8	8	9	6
S4	N	4	4	5	6	4	4
S5	N	10	9	9	10	10	10
S6	9	10	9	13	12	10	10

TABLE 7. COMPARISON BASED ON EXECUTION TIME (IN SECONDS)

Sys	AETG [31]	mAETG [31]	IPO [38]	SA [9]	GA [33]	ACA [33]	ALL Pairs [39]	MTTG
S1	N	N	N	N	N	N	0.08	0.003
S2	N	N	N	N	N	N	0.23	0.004
S3	N	N	N	N	N	N	0.45	0.02
S4	N	N	0.3	N	866	1180	5.03	0.08
S5	N	N	0.72	N	N	N	10.36	0.225
S6	N	6,001	N	10,833	6,365	7,083	23.3	0.474
S7	N	N	0.05	N	N	N	1.02	0.042
S8	N	58	N	214	22	31	0.35	0.019

TABLE 8. MTTG AND T-WAY STRATEGIES FOR TEST DATA SIZE

T-Way	IPOG [34]	ITCH [30]	Jenny [35]	Tconfig [29]	TVG [36]	GTWay [22]	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 9. EXECUTION TIME TO GENERATE TEST DATA (SECOND)

T-Way	IPOG [34]	ITCH [30]	Jenny [35]	Tconfig [29]	TVG [36]	GTWay [22]	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

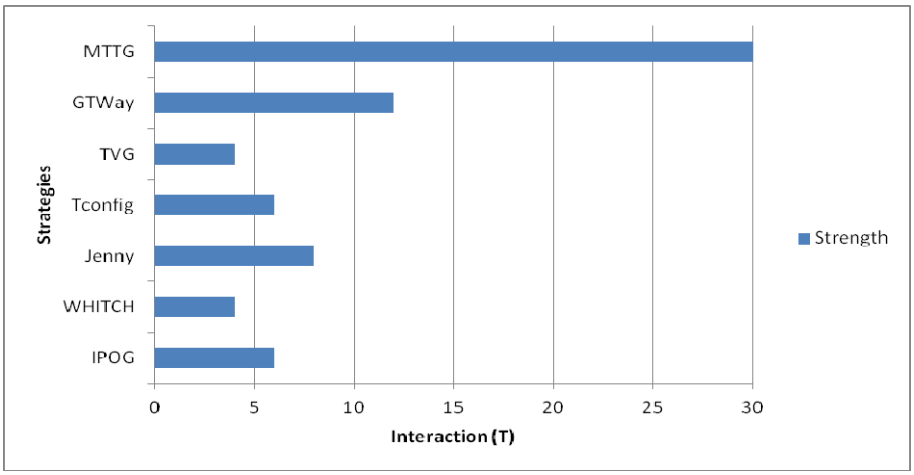

Figure 1: Maximum supportability of MTTG compare to others

D. MTTG at maximum interaction strength

An additional experiment is carried out in a fairly larger system to measure the capability of MTTG. The system consists of 30 parameters with 2 values each. MTTG is able to produce test data supporting interaction strength T up to 30 (in platform Intel P IV 1.6 GHz, 1 GB RAM). It may also be able to support higher T (T > 30) in a higher computational environment. This efficiency can be illustrated as in Figure 1. It can be illustrated that MTTG is must more efficient to support higher interaction strength. Thus MTTG can be much more efficient than any other available strategies.

V. CONCLUSION

This paper proposed a 'SET Game' based algorithm MTTG for test case generation for T-way testing. The algorithm is able to generate test data effectively compared to other algorithms. One future work is to parallelize the strategy for test data generation. Another direction for future research may include supporting non-uniform interaction. In this instance, instead of considering a uniform interaction always (i.e. T = 3, etc.), the strategy can be improved to support different interaction in single test data generation. In this case, it is necessary to evaluate both inputs and outputs to establish the necessary interactions between input values. Some of the input values may be important for a specific system, where some other may not be important at all.

REFERENCES

[1] Y. Cui, L. Li, and S. Yao, "A New strategy for pairwise test case generation," in proceedings of the 3rd international Symposium on Intelligent Information Technology Application, NanChang, China, 2009.

[2] 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 and Computer-Based Systems, Tucson, Arizona, 2007.

[3] M. I. Younis, K. Z. Zamli, and N. A. M. 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.

[4] X. Chen, Q. Gu, J. Qi, and D. Chen "Applying Particle Swarm optimization to Pairwise Testing," in proceedings of the 34th Annual IEEE Computer Software And Application Conference, Seoul, Korea, 2010.

[5] M. F. J. Klaib, S. Muthuraman, N. Ahmad, and R. 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, and 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. 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, 1997.

[8] D. M. Cohen, S. R. Dalal, A. Kajla, and G. C. Patton, "The Automatic Efficient Test Generator (AETG) System," in proceedings of the 5th International Symposium on Software Reliability engineering, Monterey, CA, USA, 1994.

[9] 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.

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

[11] 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.

[12] M. Harman and B. F. Jones. Search based software engineering. Information and Software Technology, 43(14):833–839, 2001.

[13] J. Kennedy and R. Eberhart, "Particle Swarm Optimization", in proceedings of the IEEE international Conference on Neural Networks, Washington DC, USA, 1995.

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

[15] TConfig, Available at: http://www.site.uottawa.ca/~awilliam/. (Last access date: 27th Sep. 2010).

[16] Jenny, Available at: http://www.burtleburtle.net/bob/math/. (Last access date: 27th Sep. 2010).

[17] TVG, Available at: http://sourceforge.net/projects/tvg. (Last access date: 27th Sep. 2010).

[18] W. Shitao, and W. Hao, "A Novel Algorithm for Multipath Test Data Generation", in proceedings of the 4th International Conference on Digital Manufacturing & Automation, 2013.

[19] J. L. Lions, "Ariane 5 Failure: Full Report" [Accessed on December 2010]. Available from: http://sunnyday.mit.edu/accidents.

[20] Set Game, "http://en.wikipedia.org/wiki/Set_(game)" Access Date August 21, 2014.

[21] K. F. Rabbi "Development Of Efficient T-Way Test Data Generation Algorithm And Execution Strategies," 2012

[22] M. F. J. Klaib, "Development Of An Automated Test Data Generation And Execution Strategy Using Combinatorial Approach,". 2009

[23] R. Bryce, and C. J. Colbourn, "Prioritized Interaction Testing for Pairwise Coverage with Seeding and Avoids,". Information and Software Technology Journal (IST, Elsevier), 48, 960-970. 2006.

[24] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, "Model Based Testing in Practice," in proceeding of the International Conference on Software Engineering (ICSE). 285–294, 1999.

[25] D. R. Kuhn, and V. Okum, "Pseudo-Exhaustive Testing for Software," 30th Annual IEEE/NASA Software Engineering Workshop (SEW '06). 25-27, 2006.

[26] D. R. Kuhn, and M. J. Reilly, "An Investigation of the Applicability of Design of Experiments to Software Testing," in proceedings of the 27th NASA/IEEE Software Engineering Workshop. IEEE CS Press, 69-80, 2002.

[27] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software Fault Interactions and Implications for Software Testing," in IEEE Transactions on Software Engineering 30, 418-421, 2004.

[28] J. Yan, and J. Zhang, "A Backtracking Search Tool for Constructing Combinatorial Test Suites," in Journal of Systems and Software - Elsevier, 81, 1681-1693, 2008.

[29] A. W. Williams, "Determination of Test Configurations for Pair-wise Interaction Coverage," in proceedings of the 13th International Conference on Testing of Communicating Systems. 57-74, 2000.

[30] A. Hartman, and L. Raskin, "Combinatorial Test Services". [Accessed on August 2008]. Available from: http://www.alphaworks.ibm.com/tech/cts, 2004.

[31] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design," in IEEE Transactions on Software Engineering, 23, 437-444, 1997.

[32] M. B. Cohen, C. J. Colbourn, and A. C. H. Ling, "Constructing Strength Three Covering Arrays with Augmented Annealing," Discrete Mathematics - Elsevier, 308, 2709-2722, 2008.

[33] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing," in proceedings of the 28th Annual International. Computer Software and Applications Conference (COMPSAC'04). Hong Kong, IEEE Computer Society, 72-77, 2004.

[34] Y. Lei, R. Kacker, D. Kuhn, V. Okun, V. and J. Lawrence, "IPOG/IPOD: Efficient Test Generation for Multi-Way Software Testing,". Journal of Software Testing, Verification, and Reliability, 18, 125-148, 2009.

[35] B. Jenkins, "Jenny [Online],". [Accessed on August 2011]. Available from: http://www.burtleburtle.net/bob/math/, 2003.

[36] P. J. Schroeder, E. Kim, J. Arshem, and P. Bolaki, "Combining Behavior and Data Modeling in Automated Test Case Generation," in proceedings of the 3rd International Conference on Quality Software (Qsic 2003). 247-254, 2003.

[37] S. Bestoun, A. K. Z. Zamli, and C. P. Lim, "Constructing a T-Way Interaction Test Suite Using the Particle Swarm Optimization Approach," International Journal of Innovative Computing, Information and Control (IJICIC), 8(1A): 431-452, (2012).

[38] M. Forbes, J. Lawrence, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Refining the In-Parameter-Order Strategy for Constructing Covering Arrays," NIST Journal of Research, 113, 287-297, 2008,

[39] J. Bach, "ALLPAIRS Test Generation Tool, Version 1.2.1. [Online]," [Accessed on January 2011]. Available from: http://www.satisfice.com/tools.shtml, 2004.

[40] T. Yu-Wen, and W. S. Aldiwan, "Automating Test Case Generation for the New Generation Mission Software System," in Proceeding of the IEEE Aerospace Conference. Big Sky, MT, USA, 431-437, 2000.