Ant colony algorithm to generate t-way test suite with constraints

Nuraminah Ramli¹, Rozmie Razif Othman¹, Zahereel Ishwar Abdul Khalib¹, Mohd Zamri Zahir Ahmad¹, S S M Fauzi²

¹School of Computer and Communication Engineering, Universiti Malaysia Perlis, 02600 Arau, Perlis
²Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, 02600 Arau, Perlis, Malaysia

* nuraminah@unimap.edu.my

Abstract. T-way testing is one of the testing techniques offered to generate test suites. It focuses on interactions of input parameters based on strength. Besides that, t-way is used to overcome exhaustive testing problem. Another important aspect in t-way is constraints. It forbids certain interactions of input parameters. Therefore, the final test suite contains only valid interactions. T-way testing is NP-hard problem. No single strategy can always generates the best test suite size at all time for all configurations. Thus, Const-TTSGA strategy has been developed to generate test suites. The strategy supports constraints variable strength. Const-TTSGA is a metaheuristic strategy which applies ant colony algorithm to generate the best test cases. Two types of experiments have been conducted; constraints uniform and variable strength which consists of few other configurations. Results obtained are compared to benchmarked results. Const-TTSGA outperformed other strategy for constraints uniform strength experiments except for one configuration. However, the strategy outperformed other strategies for constraints variable strength experiments.

Keyword: interaction testing, t-way testing, constraints, metaheuristic, ant colony algorithm

1. Introduction

Industry Revolution (IR) 4.0 is evolving rapidly. Smart city, smart house and many related applications were built to contribute to IR 4.0. To fulfill the technology, software applications have become more complex. There are many flow and conditions and important information to be processed. Software applications failure need to be prevented as it may cause chaos and loss in certain degree.

To prevent from software failure, software testing activities need to be given more attention. Testing technique emerged in software testing activities to generate test cases. Examples of testing techniques are boundary value analysis, equivalence partitioning and t-way testing. Each technique has its own capability in producing final test suite. As compared to the other techniques, t-way testing is focusing on interactions of two or more input parameters [1-2] because faults are not only occurred within a single parameter, but mostly within two or more input parameters [3-4]. Besides that, t-way could overcome issue of exhaustive testing which generates too many test cases. Exhaustive testing is almost impossible to be implemented because of more time and resources need to be allocated [3-6]. On the other hand, t-way generates smaller test suite size but the fault detection coverage is high [7].
T-way testing consists of three types of support interactions. They are uniform strength, variable strength and Input Output based Relationships (IOR). Many strategies have been developed to cover those three types of support interactions such as Jenny [8] and TVG [9] strategies could support uniform strength. Examples of strategies that support variable strength are Cuckoo Search [10] and Genetic Algorithm [11] while ITTDG [12] and Union [13] are the example strategies to support IOR.

Even though those strategies are useful to produce minimum number of test cases, they failed to support constraints. As software applications are becoming more complex and complicated, constraints among interaction of input parameters are important issues to be considered. In the real software system, input configurations of a system involves constraints [14]. Constraints are any forbidden interactions of two or more input parameters [15], [16]. Any combinations of the forbidden interactions are invalid and should be excluded from final test suite.

Strategies or tools that support constraints are very limited [14], [17]. In order to implement constraints, a constraints handler must be embedded in the search process, which makes the process more complex. There are existing strategies that support constraints such as General Variable Strength with Constraints (GVS_CONST) [7] and Multi-Objective Particle Swarm (MOPSO) [18]. However, the generation strategies of a test suite falls under NP-hard problem [19] which is no single strategy can generates the best test suite size for all configurations.

Due to the issue, this paper is introducing a new t-way testing strategy that supports variable strength constraints. The strategy is Constraint T-way Test Suite Generator based on Ant Colony algorithm or known as Const-TTSGA. The strategy supports uniform and variable strength as well as constraints. It applied ant colony algorithm to assist in producing minimum test suite size.

The rest of this paper is organized as follows. Section 2 explains the background study of t-way and constraints. Meanwhile, the new strategy, Const-TTSGA is presented in Section 3. In Section 4, the performance of Const-TTSGA is exhibited and discussed. Finally Section 5 concludes the paper.

2. Background study
An example of system application consists of four input parameters, A, B, C and D. Each input parameter has two values, which is represented by 1 and 2. For example, input parameter A, the values are a1 and a2. Table 1 shows a representation of input parameters and its values. To implement a 2-way testing or known as pairwise testing, each input parameter will be paired. Hence, the combinations are AB, AC, AD, BC, BD and CD. Values for each input parameter will be combined or interacted with other values, based on the combinations mentioned. It is called as tuples. Tuples generated by the combination are presented in Table 2.

| Table 1. Representation of input parameters and its values |
|-----------------|-----------------|-----------------|-----------------|
| A    | B    | C    | D    |
| a1   | b1   | c1   | d1   |
| a2   | b2   | c2   | d2   |

| Table 2. Tuples generated |
|--------------------------|
| AB   | AC   | AD   | BC   | BD   | CD   |
| a1b1 | a1c1 | a1d1 | b1c1 | b1d1 | c1d1 |
| a1b2 | a1c2 | a1d2 | b1c2 | b1d2 | c1d2 |
| a2b1 | a2c1 | a2d1 | b2c1 | b2d1 | c2d1 |
| a2b2 | a2c2 | a2d2 | b2c2 | b2d2 | c2d2 |

However, because of there are requirements that AC and BD are forbidden, the related tuples are removed from the list of tuples. Therefore, the remaining tuples are as in Table 3. These 16 tuples are the valid tuples to be included in the final test suite.
Table 3. Valid tuples after removing the constraints

| AB  | AD  | BC  | CD  |
|-----|-----|-----|-----|
| a1b1| a1d1| b1c1| c1d1|
| a1b2| a1d2| b1c2| c1d2|
| a2b1| a2d1| b2c1| c2d1|
| a2b2| a2d2| b2c2| c2d2|

The example of constraints pairwise testing is a uniform strength type of interactions support. It also can be represented by a covering array notation as in equation 1.

\[
P = CA (N, t, C)
\]  

In the equation, \( N \) is the final test suite size, \( t \) is the interaction strength. \( C \) is the value configuration and can be represented as \( V_0^{p0}, V_1^{p1}, \ldots, V_n^{pn} \) where \( p_n \) is a parameter with \( V_n \) values.

The concept of constraints uniform strength discussed can be extended to the variable strength interaction support. For variable strength, it involves multi strengths. For example, the application may have 2-way for input parameter A, B and C and also 3-way interactions for input parameter B,C,D.

Variable strength can be represented by using a covering array notation as in Equation 2.

\[
P = VCA (N, t, C, S)
\]

From the equation, \( N \) is the final test suite size and \( t \) is the dominant interaction strength. \( C \) is the value configuration and can be represented as \( V_0^{p0}, V_1^{p1}, \ldots, V_n^{pn} \) where \( p_n \) is a parameter with \( V_n \) values. \( S \) is the multi-set of disjoint covering array with strength larger than \( t \) as given in equation 1.

In this paper, constraints also can be represented as equation 3.

\[
Ct({c_{a1,b2}, c_{a-n,b-n}})
\]

where \( c_{a,b} \) refers to \( b^{th} \) value of \( a^{th} \) parameter.

3. Const-TTSGA
Const-TTSGA consists of four components as shown in Figure 1. They are Tuples Generator, Constraints Tuples Generator, Search Space Generator and Test Case Generator. Tuples Generator is used to generate tuples list based on input parameters, its respective values and strength. Meanwhile, Constraints Tuples Generator generates constraints tuples. Then, the list of tuples generates by Tuples Generator will be compared with the list of constraints tuples. Any similar tuples will be removed from the tuples list. Next, Search Space Generator generates routes to be used by a group of agent. Number of input parameters and its respective values has been used to form the routes. After the routes have been formed and the list has been updated, Test Case Generator starts to produce test cases. Finally, a test suite which contains the best test cases has been produced.
In order to search for the best test cases, a metaheuristic algorithm has been chosen because of its ability to produce smaller test suite size [20], [21]. Ant Colony algorithm which has been proven to produce a promising results for t-way test suite generation [11], [22] has been modified to suit the constraints test suite generation implementation. Figure 2 presents the flow chart of the constraints test case generator. The process is implemented for every input parameter, which has been represented as node. Each node will have edges, based on values hold by each node (i.e. input parameter). At each node, heuristic will be calculated based on tuples that can be covered. Ant colony algorithm uses a colony of ant as an agent to find the best test cases. This strategy implements random solution between exploration and exploitation. Pheromone value will be updated once the best test case has been chosen. The pheromone value will be used as guide for other ant to choose the best test case. As metaheuristic works by using fitness function, Const-TTSGA is using a fitness function that can be used by any type support interactions. The fitness function is as in [23].

Figure 1. Const-TTSGA strategy
Figure 2. Flow chart of constraints test suite generation

Initialize heuristic,
\[ \text{heuristicValue}_{\xi}(i, j) = \frac{E_{i, \text{max}} - E_{i, j} + 1}{E_{i, \text{max}} - E_{i, \text{min}} + 1} \]

Initialize pheromone, \( \tau \) value to a constant value, set ants

Construct solutions (Explore/exploit)
Set \( q_0 \) to \( 0 \leq q \leq 1 \)
Set random value, \( q \) in \([0, 1]\)

If \( q \leq q_0 \)

Explore new path,
\[ p_{i, j}(t) = \frac{[\tau_{i, j}(t)]^{\alpha}[\eta_{i, j}(t)]^{\beta}}{\sum_{h=1}^{b_{\text{best}}}[\tau_{i, h}(t)]^{\alpha}[\eta_{i, h}(t)]^{\beta}} \]

For each ant

If \( q \leq q_0 \)
Exploit path, \( \arg\max_{1 \leq h \leq b_{\text{best}}}[\tau_{i, h}(t)]^{\alpha}[\eta_{i, h}(t)]^{\beta} \)

More node?

If \( q \leq q_0 \)

Update pheromone
\[ \tau_{i, j}(t + 1) = \begin{cases} (1 - \rho)\tau_{i, j}(t) + \rho(f(t)) & \text{if } e_{i, j} \in \text{bestTest} \\ \tau_{i, j}(t) & \text{otherwise} \end{cases} \]

Calculate fitness function to find best test
\[ f(t_i) = \sum_{p=0}^{\text{program output}} W_p \]

Remove covered tuples & put the best test case in the test suite
4. Results and Discussions
Two types of experiments have been executed to see the performance of Const-TTSQA in producing test suite size. The first experiments are based on three different case studies which represents uniform strength t-way support interactions. The case studies are SPIN-S, SPIN-V and Bugzilla. Whereas, the second experiments represent variable strength t-way support interactions. Both constraints uniform and variable strength experiments are benchmarked experiments that have been used by other researchers to evaluate the performance of the strategy. Each configuration is executed for 20 independent run.

Results obtained from the experiments were compared against other existing strategies. Table 4 presents results for constraints uniform strength based experiments. Results for other strategies are taken from [18]. mAETG and AETG are computational strategies and it shows the average reported test suite size, while the other strategies (metaheuristic based strategy) shows the best test suite size. It depicts that Const-TTSQA produces the best results for all constraints uniform strength configurations except for Bugzilla system. SA based strategy produces the best results for Bugzilla system configuration.

Table 4. Results for constraints uniform strength based t-way support interactions.

| System   | CA Model | Constraints | mAETG | AETG | SA (Base) | MOPSO | Const-TTSQA |
|----------|----------|-------------|-------|------|-----------|-------|-------------|
| SPIN-S   | CA(N,2\(^1\))\(^4\)\(^5\) 2\(^3\) | 26.3 | 27.1 | 24 | 22 | **21** |
| SPIN-V   | CA(N,2\(^2\))\(^3^2\)\(^4^1\) 2\(^7\)3\(^2\) | 36 | 42.5 | 36 | 36 | **33** |
| Bugzilla | CA(2\(^4^9\)3\(^1\))\(^2^4^3\) | 25.0 | 24.9 | **17** | 25 | 20 |

Meanwhile, Table 5 shows results of constraints variable strength based experiments. Results for other strategies are from [7] and present the best reported test suite size. From this table, only Const-TTSQA strategy applies metaheuristic search technique while other strategies are computational strategies. Cell with bold font is the best result or the smallest test suite size for the respective configuration. Cell with ‘NA’ means no result is available in the literature. Const-TTSQA produces the best result for all constraints variable strength configurations. It is proven that metaheuristic based strategy generates better results [20]. Among all strategies, PICT generates the worst results for all configurations. In general, test suite size generated for constraints configuration will be smaller than configuration without constraints. However, IPOG reported that the results are not affected by constraints or without constraints configurations.

Table 5. Results for constraints variable strength based t-way support interactions.

| CA Model | Constraints | GVS_CONST | GTWAY | IPOG  | PICT  | Const-TTSQA |
|----------|-------------|-----------|-------|-------|-------|-------------|
| VCA(N,3\(^3\), CA(4,3\(^3\))) | Ct() | 118 | 125 | 119 | 900 | **105** |
| VCA(N,3\(^3\), CA(4,3\(^3\))) | Ct({c_{1,2,c_{4,1,c_{7,1}}}, [c_{2,1,c_{6,2},c_{7,2}}], [c_{3,1,c_{4,1,c_{5,3}}}, [c_{2,3,c_{4,2,c_{6,3}}}, [c_{2,2,c_{4,1,c_{7,3}}}, [c_{1,1,c_{2,2,c_{5,3,c_{6,2}}}}]) | 112 | NA | 116 | 765 | **101** |
| VCA(N,3\(^3\), CA(4,4\(^3\))) | Ct() | 365 | 377 | 352 | 5328 | **326** |
| VCA(N,3\(^3\), CA(4,4\(^3\))) | Ct({c_{3,2,c_{5,3}}}, [c_{2,2,c_{6,4,c_{7,4}}}, 344 | NA | 355 | 4958 | **320** |
5. Conclusion

In this paper, Const-TTSGA strategy has been introduced to generate test suite size. Const-TTSGA is focusing on constraints and variable strength support interactions. The strategy is a metaheuristic based strategy, which applies ant colony algorithm to assist in generating the minimum test suite size. The performance of the strategy is evaluated by implementing two types of experiments. The first type is to evaluate on constraints uniform strength, where three experiments have been conducted. Whereas, the second type of experiments is conducted to evaluate on constraints variable strength. Three experiments with six different configurations have been conducted. Results obtained from the experiments are compared to other existing strategies that support the respective support interactions. Const-TTSGA generates the best test suite size for constraints uniform strength except for Bugzilla case study. Besides that, it shows very good results where the strategy generates the best test suite size for all constraints variable strength. For the future works, experiments for higher strength and configurations will be conducted to see the performance of Const-TTSGA.

Acknowledgments

The author would like to acknowledge the support from the Fundamental Research Grant Scheme (FRGS) under a grant number of FRGS/1/2018/ICT01/UNIMAP/02/1 from the Ministry of Education Malaysia.

References

[1] R. R. Othman and K. Z. Zamli, “T-Way Strategies and Its Applications for Combinatorial Testing,” Int. J. New Comput. Archit. Their Appl., vol. 1, no. 2, pp. 459–473, 2011.

[2] N. Ramli, R. R. Othman, Z. I. Abdul Khalib, and M. Jusoh, “A Review on Recent T-way Combinatorial Testing Strategy,” MATEC Web Conf., vol. 140, no. 1016, pp. 1–6, 2017.

[3] D. R. Kuhn, R. N. Kacker, and Y. Lei, Introduction to Combinatorial Testing. Boca Raton: CRC Press, 2013.

[4] 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 International Symposium and Workshop on Engineering of Computer Based Systems, 2007, pp. 549–556.

[5] R. R. Othman, K. Z. Zamli, and S. M. S Mohamad, “T-Way Testing Strategies: A Critical Survey and Analysis,” Int. J. Digit. Content Technol. Its Appl., vol. 7, no. 9, pp. 222–235, 2013.

[6] R. C. Bryce, Y. Lei, D. R. Kuhn, and R. Kacker, “Combinatorial Testing,” in Handbook of Research on Software Engineering and Productivity Technologies, IGI Global, 2010, pp. 196–208.

[7] R. R. Othman, N. Khamis, and K. Z. Zamli, “Variable Strength t-way Test Suite Generator with Constraints Support,” Malaysian J. Comput. Sci., vol. 27, no. 3, pp. 204–217, 2014.

[8] J. B. “Jenny Test Tool.”

[9] J. Arshem, “Test Vector Generator Tool (TVG),” 2010. [Online]. Available: http://sourceforge.net/projects/tvg. [Accessed: 20-Aug-2001].

[10] A. B. Nasser, A. A. Alsewari, and K. Z. Zamli, “Learning Cuckoo Search Strategy for t-way
Test Generation,” in *Computing, Analytics and Networks*, vol. 805, Rajnish Sharma, A. Mantri, and S. Dua, Eds. Springer Singapore, 2018, pp. 97–110.

[11] 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, 2004. COMPSAC 2004.*, 2004, pp. 72–77.

[12] R. R. Othman and K. Z. Zamli, “ITTDG : Integrated T-way Test Data Generation Strategy for Interaction Testing,” *Sci. Res. Essays*, vol. 6, no. 17, pp. 3638–3648, 2011.

[13] P. J. Schroeder, “Black-box Test Reduction using Input-Output Analysis,” Illinois Institute of Technology, 2001.

[14] B. S. Ahmed, K. Z. Zamli, W. Afzal, and M. Bures, “Constrained Interaction Testing: A Systematic Literature Study,” *IEEE Access*, vol. 5, 2017.

[15] L. Yu, F. Duan, Y. Lei, R. N. Kacker, and D. R. Kuhn, “Constraint handling in combinatorial test generation using forbidden tuples,” *2015 IEEE 8th Int. Conf. Softw. Testing, Verif. Valid. Work. ICSTW 2015*, pp. 1–9, 2015.

[16] J. Petke, “Constraints: The Future of Combinatorial Interaction Testing,” in *8th International Workshop on Search-Based Software Testing, SBST 2015*, 2015.

[17] S. K. Khalsa and Y. Labiche, “An orchestrated survey of available algorithms and tools for combinatorial testing,” in *25th International Symposium on Software Reliability Engineering, ISSRE*, 2014, pp. 323–334.

[18] B. S. Ahmed, L. M. Gambardella, W. Afzal, and K. Z. Zamli, “Handling constraints in combinatorial interaction testing in the presence of multi objective particle swarm and multithreading,” *Inf. Softw. Technol.*, vol. 86, pp. 20–36, 2017.

[19] C. Nie and H. Leung, “A Survey of Combinatorial Testing,” *ACM Comput. Surv.*, vol. 43, no. 2, pp. 1–29, 2011.

[20] K. Z. Zamli, B. Y. Alkazemi, and G. Kendall, “A Tabu Search Hyper-Heuristic Strategy for T-way Test Suite Generation,” *Appl. Soft Comput. J.*, vol. 44, pp. 57–74, 2016.

[21] A. A. Alsewari and K. Z. Zamli, “Design and Implementation of a Harmony-Search-Based Variable-Strength T-way Testing Strategy with Constraints Support,” *Inf. Softw. Technol.*, vol. 54, no. 6, pp. 553–568, 2012.

[22] X. Chen, Q. Gu, A. Li, and D. Chen, “Variable Strength Interaction Testing with an Ant Colony System Approach,” in *Asia-Pacific Software Engineering Conference, APSEC, 2009*, pp. 160–167.

[23] N. Ramli, R. R. Othman, and M. S. A. Rashid Ali, “Optimizing Combinatorial Input-Output Based Relations Testing using Ant Colony Algorithm,” in *3rd International Conference on Electronic Design (ICED), August 11-12, 2016, Phuket, Thailand*, 2016, pp. 586–590.