A Tuned Version of Ant Colony Optimization Algorithm (TACO) for Uniform Strength T-way Test Suite Generator: An Execution’s Time Comparison

Mohd Zamri Zahir Ahmad, Rozmie Razif Othman, Mohd Shaiful Aziz Rashid Ali, Nuraminah Ramli, Mohd Wafi Nasrudin and Ahmad Ashraf Abdul Halim
Faculty of Electronic Engineering Technology, Universiti Malaysia Perlis, Pauh Putra, 02600 Arau, Perlis, Malaysia

E-mail: zamri.zahir@yahoo.com, rozmie@unimap.edu.my

Abstract. Software testing is one of important phase in software development. The capabilities of t-way testing to cater bugs due to interactions while reducing the test suite size compare to exhaustive testing has been proven in past decades. However, the execution’s time of the t-way strategy also should be given attention as it could increase the productivity of the testing phase. Thus, this paper proposed a tune version of ant colony optimization algorithm (TACO). TACO is metaheuristic strategy where it adopts ant colony optimization in generating test suites. As further improvement, TACO also integrated with fuzzy logic to dynamically select amount of ant in the algorithm. TACO able to supports uniform strength t-way testing. Experiment result shows that TACO produce a remarkable result of test suite size and execution’s time compared to other strategy for uniform strength t-way testing.

Keywords : Combinatorial testing, T-way testing, Ant Colony Optimization.

1. Introduction
Software testing is one of the stages involve in software development life cycle (SDLC). It is one of most crucial in developing a software as to ensure the correctness and adhere the quality of the software. Hence, it is necessary to test the software in several aspects in order to minimize fault before releasing the end user. By selecting a proper test plan that offer minimal execution time, might increase the productivity of the software company too.

Fault due to interactions is one of crucial errors type that need to be reveal, as any failure between these interactions, will affect the main functionalities of the system [1][2]. Even though, an exhaustive testing is good approach to detect all possible combinations fault, it also consumes a longer time while in most of case it is practically impossible to be implement which for larger and complex system [3]. As to avoid combinatorial explosion, there are findings in literature suggest that sampling strategy as t-way interaction testing) is effective to minimize the number of test cases by caters faults due to interactions in a most typical software system [4][5]. However, t-way testing is fall in NP Hard problem where no single strategy can claim it always generates best test suite size for all configurations [4][6][7].

Ant colony optimization algorithm (ACO) is one of metaheuristic strategy that been proposed in earlier research to cater problem of combinatorial optimization. ACO is an algorithm that mimics of ant...
colonies in searching for the food via shortest path. ACO adopts maximum iterations, search space size, evaporation rate, pheromone and heuristic effect as the controlling parameter in searching for the optimum test suite [8]. Technically, ACO utilize a set of ant agent for every iteration in searching of test case while eventually, every ant agent will form a single test case. However, only the best test case will be pick to be pushed in final test suite. One of the success factor of ACO in producing smaller test suite size is depending on the number of ants. Increasing number of ants would increase the probability of producing optimum test suite size, yet also increasing the execution’s time [9].

While useful, conventional ACO tend to have a slow execution’s time while producing the best test cases. This is due to the number of ant is set to a fixed amount for every iteration of the configurations, thus making the over utilized number of ants in certain iterations affecting the execution’s time. Therefore, in this paper we proposed a tuned version of ACO algorithm (TACO) that support uniform strength of t-way test suite generation strategy. TACO is also capable to enhance the execution time while preserved the effectiveness of ACO in producing the minimal test suite size. Briefly, TACO will balance the amount of ants for every iteration by considering iteration’s phase and tuples coverage. TACO dynamically tune number of ants depending on iterations and tuple coverage of the ACO. A fuzzy logic mamadani-type been utilized in TACO in order to accomplished the dynamic selection of ants based on determined fuzzy inference system (FIS) criteria.

2. Covering Array with Problem Definition
As alternative approach to cater optimization problem of combinatorial explosion due to exhaustive testing, covering array (CA) is proposed to gather all possible test interactions that have high potential to uncover faults [10]. In brief, CA is a minimized array that represent all interactions of the component, which will be used as a final test suite. CA denoted as CA (N; t, p, v), where N is a size of array with v values, such that every Nx t sub-array contains all ordered subsets from v symbol of size t at least once. The strength of the array is represented by t. Nevertheless, there are case that the number of components values is not always uniform where it may vary according to the practice. Hence, this case can be solve by using Mixed Covering Array, MCA (N; t, k, (v¹, v², . . . , v¹)) or in simplified form as MCA (N; t, k, v¹).

To demonstrate uniform strength of t-way testing, let us consider a simple example of a system for online shopping (e-Shop). In this system, buyer that placed an order of the product to the seller, will then makes the payment and choose preferred shipping courier. Buyer also could apply any voucher if available to get discount before making payment. Seller then will verify all details before ship out the products to the buyer.

| Buyer, (A) | Seller, (B) | Payment option, (C) | Shipping courier, (D) | Voucher provider, (E) |
|------------|------------|---------------------|-----------------------|----------------------|
| Peter, (a¹) | SKFurniture, (b¹) | Cash, (c¹) | J&T, (d¹) | Seller, (e¹) |
| Dwayne, (a²) | GadgetPlanet, (b²) | Online banking, (c²) | PosLaju, (d²) | Apps discount, (e²) |
| e-Wallet, (c³) | NinjaVan, (d³) |

The system of the e-Shop component is a sequence less parameter and have its own values as listed in Table 1. Each components marks with an alphabet notation to ease the further explanations. There are 5 parameters (p=5) involve and each parameter have its own possible values (v¹=2, v²=2, v³=3, v⁴=3, v⁵=2). Since if the system need to run for an exhaustive test, a total of 72 test case (i.e. 2x2x3x3x2) need
to be execute when full interaction strength, $t$ is applied ($t=5$). However, the interaction strength, $t$ also can relax to 2, where there are only 2 input parameters interacting and faults are expected occur between these interactions. Table 2 shows the possible combinations (or called as tuples) produces when $t$ is set to 2. As listed, there are 57 tuples generated for the mentioned configurations that must be covered by the final test cases.

Table 2. Interacting Tuples.

| AB | AC | AD | AE | BC | BD | BE | CD | CE | DE |
|----|----|----|----|----|----|----|----|----|----|
| a1b1 | a1c1 | a1d1 | a1e1 | b1c1 | b1d1 | b1e1 | c1d1 | c1e1 | d1e1 |
| a1b2 | a1c2 | a1d2 | a1e2 | b1c2 | b1d2 | b1e2 | c1d2 | c1e2 | d1e2 |
| a2b1 | a2c3 | a2d3 | a2e1 | b2c3 | b2d3 | b2e1 | c2d3 | c2e1 | d2e1 |
| a2b2 | a2c2 | a2d2 | a2e2 | b2c2 | b2d2 | b2e2 | c2d2 | c2e2 | d2e2 |
| a2c3 | a2d3 | b2e3 | b2d3 | c2d3 | c2e2 | d2e2 |
| c3d1 |       |       |       |       |       |       |
| c3d2 |       |       |       |       |       |       |
| c3d3 |       |       |       |       |       |       |
| 4   | 6   | 6   | 4   | 6   | 6   | 4   | 9   | 6   | 6 |

Total Tuples = 57

Table 3 shows each of generated test case with it covered tuples by using TACO t-way testing. Only 11 test cases out of 72 test cases of exhaustive test suite are needed to be execute and it is adequate to cover all tuples produces in Table 2. This mean, the total reduction achieved is over 84% compared to the exhaustive test suite. The full notation of CA for this example can be written as, MCA (11; 2, 2332).

Considering for a larger and complex system, increasing the parameter and values would increase the exhaustive test case, therefore increases the testing cost and time. For example, a system with 30 input with 2 possible value each will resulted 1,073,741,824 (i.e. $2^{30}$) test cases need to be executed. Obviously, this method of exhaustive testing is not efficient and need for other testing method to overcome the problems. T-way testing is one of the solution approach design to cater with the problems with a systematic sampling. Thus, the reduction of 84% as shows in previous example leave a significance marks to cater the problem in software combinatorial testing.
3. Background Studies

In past studies, T-way testing has been generalized in three main areas, which is strategy approach, search technique and supported interaction [11]. There are two strategy approach of T-way testing which is one-parameter-at-a-time (OPAT) and one-test-at-a-time (OTAT) [12] [13]. OPAT strategy adopt a horizontal as well as vertical parameters extension in producing the test cases. The strategies start generating test suite from smallest t-combination and add the parameters horizontally per iterations until all parameter covered [14]. While for OTAT, it generates single test case per iteration that can cover most uncovered tuples. The second area is searching technique. There are two searching technique been used which is computational and metaheuristic. Computational techniques are the conventional derived from algebraic method but offers great flexibility to the configurations setting. It applies greedy method in constructing test cases so that be able to cover the most uncovered tuples. However, there are some disadvantages of this method. Generated test case from greedy method often not optimal as they tend to stuck in local optima. Other than that, when the numbers of combination are too large, they explicitly enumerating all possible combinations to be covered in the search space [4].

Metaheuristic is another searching technique that has been on the rise recently. This technique often begins the searching by population of random solutions (i.e. test cases). Then, a series of process will be applied in searching for better solutions. The process the will iterated until the selected better solutions covers all the uncovered tuples [15]. It is reported that metaheuristic searching technique has proven to generates an optimum test suite size compare to others available technique despite it consume longer time. For supported interaction, there are three types of interaction to construct test case, which is uniform strength, variable strength and input output based relation.

| Test Case # | Test Case | Tuples Covered | Total Covered Tuples |
|-------------|-----------|----------------|----------------------|
| 1           | a₁b₂c₃d₂e₁ | (a₁b₂), (a₁c₃), (a₁d₂), (a₁e₁), (b₂c₃), (b₂d₂), (b₂e₁), (c₃d₂), (c₃e₁), (d₂e₁) | 10                   |
| 2           | a₁b₁c₁d₁e₂ | (a₁b₁), (a₁c₁), (a₁d₁), (a₁e₂), (b₁c₁), (b₁d₁), (b₁e₂), (c₁d₁), (c₁e₂), (d₁e₂) | 10                   |
| 3           | a₂b₁c₂d₂e₂ | (a₂b₂), (a₂c₂), (a₂d₂), (a₂e₂), (b₂c₂), (b₂d₂), (b₂e₂), (c₂d₂), (c₂e₂), (d₂e₂) | 9                    |
| 4           | a₂b₁c₂d₃e₁ | (a₂d₃), (a₂e₁), (b₂d₃), (b₂e₁), (c₂d₃), (c₂e₁), (d₃e₁) | 7                    |
| 5           | a₂b₂c₁d₂e₁ | (a₂b₂), (a₂c₁), (b₂c₁), (b₂d₂), (c₁e₁) | 5                    |
| 6           | a₁b₂c₂d₁e₂ | (a₁c₂), (b₂c₂), (b₂d₁), (b₂e₂), (c₂d₁) | 5                    |
| 7           | a₂b₁c₃d₂e₂ | (a₂c₃), (b₁c₃), (c₁e₂) | 3                    |
| 8           | a₂b₂c₃d₁e₁ | (a₂d₁), (c₃d₁), (d₁e₁) | 3                    |
| 9           | a₁b₁c₁d₃e₂ | (a₁d₁), (c₁d₃), (d₁e₂) | 3                    |
| 10          | a₂b₁c₁d₃e₁ | (c₁d₃) | 1                    |
| 11          | a₁b₂c₁d₃e₁ | (b₂d₃) | 1                    |

**Table 3. Generated Test Case with Covered Tuples.**

Total Covered 57
ACO algorithm is a metaheuristic algorithm where it is inspired from the real foraging behavior of ants, travelling in search for food from the nest via the shortest path available [16]. In 2019, Nuraminah designed a new ACO algorithm to generate test suite that support uniform strength, known as TTSGA [17]. TTSGA utilize pheromone and heuristic values in the search process. It also applies a fixed number of ants for every iteration in searching for the best test cases. The algorithm starts with accepting input parameter, values and strength. Then a tuple generator is assign to generate tuples based on the input configurations, which then is stored in tuple list. While generating test case using test case generator, the covered tuples in tuples list will be removed. The process iterated until all tuples has been covered by the generated test cases. The best test case then will be pushed into final test suite.

4. A Tuned Version of ACO (TACO)

TACO is a newly inspired metaheuristic algorithm from ACO algorithm where it adopts OTAT searching approach. Different from TTSGA, numbers of ant in TACO will be dynamically choose based on iteration’s phase and the tuple coverage by using fuzzy logic mamdani-type. Figure 1 the general idea of TACO.

Figure 1. The general idea of TACO.

Figure 2 shows the main function of TACO. It consists of four main parts, and these factors drive the effectiveness of TACO. TACO algorithm start with receiving input from the software tester. It need number of parameters with its values and the interaction strength. Ant Colony Path Generator will start to generate virtual nodes after receiving inputs. The input parameter will be represented as the virtual nodes, while the values of the parameter is represented by the branch of each node. The generated route will be use by ants in searching for the optimum route (i.e. constructing the test case) which can cover most uncovered tuples. In the meantime, tuple generator will retrieve input configurations and start to generate a list of tuples. The list of tuples then will be stored in Tuple List.
Figure 2. Main Framework of TACO

Ant Fuzzy Controller then will dynamically choose number of ant based on iteration phase and remaining uncovered tuples. The rules apply is that for early and near end iteration phase, only small number of ant need to be choose. This is due to, at early iteration, there are still many possible solutions available so, the greater numbers of ant will result less significance improvement to the algorithm. Same as at the near end iterations phase, only few tuples left to be covered means small numbers of ant is adequate to be selected. In the middle of the iteration phase, a large amount of ant is set to be choose as this will help finding the best test case that can cover most of the tuples. The test case generator will generate test cases based on number of ants while only the best test case (which can cover most of uncovered tuples) will be selected to be pushed in final test case. Each of tuples that covered by the best test cases will be removed from tuple list. The process will iterate until all tuples have been covered.

5. Result & Discussion
As stated in previous sections, the TACO strategy is to minimize the execution’s time of generating the t-way test suite size. In this experiment, both TACO and TTSGA are implemented in java (JDK 1.8) programming language using NetBeans software. The running environment for the experiments is compromises of a desktop PC with Windows 7, 3.20 GHz Intel(R) Core(TM) i5-4460 CPU with 4 GB of RAM. To evaluate the performance of TACO, three different experiments were conduct to observe the significance results.

a) First experiment conducted was for fixed parameters with varies interactions strength and values. The configuration consists of 7 parameters with varied interaction strength, from 2 to 5. Each interaction strength consists of 4 values, from 2 to 5, CA \((N; t, v^7)\).

b) Second experiment conducted is for fixed strength with varies parameters and values. The configuration consists of interaction strength of 3 and varied parameters from 4 to 6. Each parameter consists of 4 values, 3 to 6, CA \((N; 3, v^p)\).

c) Thirds experiment conducted is for MCA where multi parameter, value and interactions strength is applied and tested.
Table 4. Generated Test Suite size for CA \((N; t, v)\).

| \(t\) | \(v\) | TTSGA \(\text{Execution’s Time (milisecond)}\) | TTSGA \(\text{Test Suite Size}\) | TACO \(\text{Execution’s Time (milisecond)}\) | TACO \(\text{Test Suite Size}\) |
|-------|-------|--------------------------------|----------------|--------------------------------|----------------|
| 2     | 2     | 2387                          | 12             | 1825                          | 8              |
| 3     | 11060 | 27                            | 9765           | 28                            |
| 4     | 22839 | 50                            | 208526         | 47                            |
| 5     | 46551 | 83                            | 32027          | 84                            |
| 3     | 14524 | 24                            | 9141           | 22                            |
| 3     | 65551 | 97                            | 45272          | 96                            |
| 4     | 228853| 242                           | 154908         | 240                           |
| 5     | 743372| 484                           | 478515         | 475                           |
| 4     | 2     | 42791                         | 48             | 23868                         | 47             |
| 3     | 287556| 289                           | 224999         | 282                           |
| 4     | 1712898| 964                        | 1372366        | 973                           |
| 5     | 8419241| 2396                       | 5458516        | 2417                          |
| 5     | 44991 | 76                            | 31731          | 79                            |
| 3     | 750517| 710                           | 618291         | 728                           |
| 4     | 8015122| 3318                       | 5394592        | 3316                          |
| 5     | 66965602| 10409                      | 41140453       | 10457                         |

Table 5. Generated Test Suite size for CA \((N; 3, v^p)\).

| \(p\) | \(v\) | TTSGA \(\text{Execution’s Time (milisecond)}\) | TTSGA \(\text{Test Suite Size}\) | TACO \(\text{Execution’s Time (milisecond)}\) | TACO \(\text{Test Suite Size}\) |
|-------|-------|--------------------------------|----------------|--------------------------------|----------------|
| 4     | 2     | 905                           | 13             | 952                           | 11              |
| 3     | 7207  | 51                            | 6474           | 45                            |
| 4     | 36083 | 112                           | 19344          | 121                           |
| 5     | 67033 | 234                           | 66955          | 231                           |
| 5     | 2     | 2262                          | 16             | 2153                          | 18              |
Table 6. Generated Test Suite size for MCA ($N$, $t$, $v$).

| Configurations | TTSGA | TACO |
|----------------|-------|------|
|                | Execution’s Time (milisecond) | Test Suite Size | Execution’s Time (milisecond) | Test Suite Size |
| MCA (n;2,51 31 21) | 3463 | 24 | 5896 | 24 |
| MCA (n;3,41 21 31 51) | 83788 | 161 | 84256 | 164 |
| MCA (n;4,31 51 41 21) | 492431 | 392 | 337725 | 385 |
| MCA (n;5,51 41 31) | 5734211 | 2650 | 3864185 | 2705 |

Result in Table 4, 5 and 6 shows TACO produces a competitive result compared to TTSGA. In Table 4 shows that on lower interaction strength, TACO produce a better test suit size. Nevertheless, for higher interaction strength and values, TACO performance is slightly below than TTSGA. Table 5 shows, TACO still manages to generate minimum test suite size for 7 out of 12 testing conducted. While in Table 6, TACO produce a slightly higher test suite size for configurations 2 and 4. It is notable that, TACO perform better result for lower interaction strength with higher parameter. Yet, TACO shows a remarkable improvement of all testing configurations in term of execution’s time. For example, in table 4, at $t = 5$ and $v = 5$, TACO offered 38% reduction of execution’s time compared to TTSGA.

6. Conclusion and Future Works
This paper highlights the newly design of ACO algorithm called as TACO. TACO produces a competitive test suite size compared to TTSGA. Apart from that, TACO execution’s time also shows a remarkable reduction compare to TTSGA as well. Further modification can be made in TACO in order to produce a better test suite size. Tuning the searching mechanism technique of exploration and exploitation by using fuzzy logic algorithm would probably reduce test suite size. This is due to current implementation of ACO searching mechanism only depending on fair randomness technique.

Acknowledgement
The author would like to acknowledge the support from 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] 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.

[3] X. Qu and M. B. Cohen, “A Study in Prioritization for Higher Strength Combinatorial Testing,” Proc. - IEEE 6th Int. Conf. Softw. Testing, Verif. Valid. Work. ICSTW 2013, no. March 2013, pp. 285–294, 2013, doi: 10.1109/ICSTW.2013.40.

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

[5] 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.

[6] M. S. A. Rashid Ali, R. R. Othman, Z. R. Yahya, and M. Z. Zahir, “A Modified Artificial Bee Colony Based Test Suite Generation Strategy for Uniform T-Way Testing,” IOP Conf. Ser. Mater. Sci. Eng., vol. 767, no. 1, 2020.

[7] M. Z. Z. Ahmad, R. R. Othman, and M. S. A. Rashid Ali, “Sequence Covering Array Generator (SCAT) for Sequence Based Combinatorial Testing,” Int. J. Appl. Eng. Res., vol. 11, no. 8, pp. 5984–5991, 2016.

[8] X. Chen, Q. Gu, X. Zhang, and D. Chen, “Building prioritized pairwise interaction test suites with ant colony optimization,” Proc. - Int. Conf. Qual. Softw., pp. 347–352, 2009.

[9] M. Z. Zahir Ahmad, R. R. Othman, M. S. A. Rashid Ali, and N. Ramli, “A Self-Adapting Ant Colony Optimization Algorithm Using Fuzzy Logic (ACOF) for Combinatorial Test Suite Generation,” in IOP Conference Series: Materials Science and Engineering, 2020, vol. 767, no. 1.

[10] K. Z. Zamli, B. S. Ahmed, and K. Z. Zamli, “A Review of Covering Arrays and Their Application to Software Testing,” J. Comput. Sci., vol. 7, no. 9, pp. 1375–1385, 2011.

[11] 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, pp. 1–6, 2017.

[12] R. R. Othman, K. Z. Zamli, and L. E. Nugroho, “General Variable Strength T-Way Strategy Supporting Flexible Interactions,” Maejo Int. J. Sci. Technol., vol. 6, no. 03, pp. 415–429, 2012.

[13] A. B. Nasser, F. Hujainah, A. A. Al-Sewari, and K. Z. Zamli, “An improved jaya algorithm-based strategy for t-way test suite generation,” Adv. Intell. Syst. Comput., vol. 1073, pp. 352–361, 2020.

[14] 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.

[15] A. A. B. Homaid, A. A. Alsewari, A. K. Alazzawi, and K. Z. Zamli, “A Kidney Algorithm for Pairwise Test Suite Generation,” Adv. Sci. Lett., vol. 24, no. 10, pp. 7284–7289, 2018.

[16] M. Dorigo and T. Stutzle, Ant Colony Optimization: Overview and Recent Advances, Handbook o., vol. 146, no. Chapter 8. Springer International Publishing AG, 2010.

[17] N. Ramli, “The Design of T-way Test Suite Generator Supporting Uniform Strength , Variable Strength and Input-Output Based Relations,” Universiti Malaysia Perlis, 2019.