Implementation of Sine Cosine Algorithm (SCA) for Combinatorial Testing

J M Altmemi1,2, R R Othman1, R Ahmad1 and A S Ali1

1School of Computer and Communication, Universiti Malaysia Perlis, Kampus Pauh Putra, Arau, Perlis, Malaysia
2Iraq University College, Basra, Iraq

jalalaltmemi@gmail.com

Abstract. Before being released to the market, software should be screened to ensure that the quality assurance measurement goals have been attained. To achieve this, one of the types of testing sorts is combinatorial testing (CT) aimed at discovering the faults that occur by interacting with the software. A minimization strategy for test cases is indeed important for optimizing test cases and reducing time. As NP hard (where NP is a non-deterministic polynomial) is the problem of generating the minimum test suite of combinatorial interaction testing (CIT). this paper discusses the implementation, and validation of an efficient strategy for t-way testing. The main contribution of the sine cosine algorithm SCA is to show that the strategy was sufficiently competitive as compared to other strategies in terms of the generated test suite size. Unlike most paper. The main contribution of SCA is to show the generation of test data for a high coverage strength (t < 12).

1. Introduction

Computer use enters every corner of social life and is the major driver of economic and social advancement. Software has become a main problem influencing social and economic development as the centre of information technology. Several software testing techniques have been suggested and examined to enhance the quality of software. Different test techniques are aimed at varying kinds of faults and can be efficient for different test scenarios. For example, mutation testing realizes the concept of using artificial flaws to encourage testing. Many empirical studies demonstrate that an efficient interaction space, e.g. targeting detection of faults, only constitutes a tiny percentage of the general interaction space in real software systems[1],[2]. In addition, there is a comparatively tiny amount of variables engaged in these efficient interactions, of which 4 to 6 are generally the upper boundaries.[3] Mutation is typically used as a manner of assessing the adequacy of test suites, guiding test case generation and supporting experimentation. But mutation has reached maturity and is gaining popularity gradually in both academia and industry[4].

Also, metamorphic testing (MT) is a technique conceived to alleviate the oracle problem. It is based on the idea that often it is simpler to reason about relations between outputs of a program, than it is to fully understand or formalise its input output behavior. But the overall effectiveness of MT is still lacking or poor [5]. Furthermore, some previous MT experiments have yielded contradictory results.
In this article we are focusing on a promising test which is combinatorial testing (CT), also known as combinatorial interaction testing (CIT)[6, 7]. CT has been shown to be efficient in detecting system interaction flaws. CT is helpful in identifying certain flaws because CT can be regarded as offering a kind of software testing shortcut. However, it has its own pitfalls, like not testing all possible combinations of parameters. If the parameters and their values are not correctly chosen, CT's ability to detect defects will be reduced[6], the aforementioned test also called In this paper we implement the sine cosine algorithm for CT to enhance the performance of the testing method.

Recent attempts to improve CT have concentrated on the implementation of meta-heuristic algorithms as the foundation of t-way approaches (where t indicates interaction strength) by formulating interaction testing as an optimization issue (e.g. genetic algorithms (GA)[8], particle swarm optimization (PSO)[9], harmony search algorithm (HS)[10], ant colony algorithm (ACO)[11], simulated annealing (SA)[12], flower pollination algorithm[13], and cuckoo search (CS)[14]). The implementation of these meta-heuristic strategies seems to be efficient in achieving alternates of excellent quality. As recorded in t-way testing benchmarking studies[15, 16], the sine cosine algorithm outperforms all the aforementioned algorithms[17].

The rest of this paper consists of

- Introduction
- Combinatorial testing
- Sine cosine algorithm
- Result and discussion
- Conclusion

2. Combinatorial testing

The purpose of functional testing and specification-based testing is to define test cases that can exercise an application's functional requirements by systematically choosing representative combinations of parameter values. In reality, a software system generally relies on many parameters, like input parameters, state variables, and environments for the configuration. This makes it difficult to test the system taking into consideration all possible parameter value combinations. In the literature various techniques have been suggested over the last 35 years to tackle this problem, such as equivalence partitioning, boundary value analysis, as well as combinatorial testing[18]. To be used in software testing, they select representative and appropriate combinations of parameter value combinations. In this paper we use CT to solve the software testing issue. CT usually involves data intensive systems, system settings, and GUI-based testing for example[18-23]. CT is based on the assumption that a subset of parameter combinations is considered instead of testing all possible combinations of parameters, which satisfies some predefined combinations strategies. That can generate combinations of parameters that allow them to cover all parameter pairs (triples, etc.) called t-way combinations. Therefore, in t-way testing for example (2-way, 3-way) it is necessary to identify test suites so that all their value combinations are covered by at least one test case given any t input parameters. The names of these test suites called cover arrays[18, 24].

2-way testing (also called pairwise testing) is the most prevalent form of t-way testing. In 2-way testing the test suite must contain test cases so that each combination of their values is covered by at least one test case given each of the two input parameters. For example, let us think that we have a system with four input parameters (A, B, C and D), each with an integer varying from 0 to 50. In order to test such a system using all input value combinations, it is necessary to call system call 504 times. Alternatively, we simply assume all combinations of pairs of input values by implementing pair testing, for example, [A, B], [A, C], [A, D], [B, C], [B, D], [C, D] etc. Through this method, we are reducing the number of systems calls to about 502, which is two orders of magnitude less than the exhaustive generation of all combinations. To clarify the concept, let's consider a system with k parameters, each with v values. We need v^k test cases to test such a system for all input combinations, one for each possible combination, whereas the number of test cases increases logarithmically in k [25, 26] with pairwise testing. The fundamental rationale for CT is that not each parameter leads to every fault, and that interactions among a several parameters often cause a fault[27]. Empirical experience[28] has
shown that CT can become very efficient in decreasing the size of test cases without wasting too much coverage of application code for example [29-31]. In fact, fault revealing suite capabilities (2-way and 3-way testing can capture 50-75% of faults [31-34]). CT issues have been addressed by several researchers through the development of different strategies based on many different approaches or categories, including pure computational approaches (e.g. PICT[35, 36], density[37], TVG[38], and ITCH[39]) and pure search approaches (e.g. SA[40], ACA[41], and HSS[42]). This article aims to review the works involved in implementing the SCA.

3. Sine cosine algorithm

The SCA is a meta-heuristic algorithm based on the population [5]. Accordingly, as its name suggests, the SCA exploits the functions of sine and cosine to update the positions of its population. Each position is a vector handle. The positions are updated to be specific depending on:

\[ x_i^{(t+1)} = x_i^t + r_1 \sin (r_2) \times |r3p_i^t - x_i^t|, \ r_4 < 0.5 \]  
\[ x_i^{(t+1)} = x_i^t + r_1 \cos (r_2) \times |r3p_i^t - x_i^t|, \ r_4 \geq 0.5 \]

In the above equations, \( x_i^{(t)} \) refers to the current solution's position in the \( i \)th dimension and the \( i \)th iteration; \( r_1, r_2, r_3 \) and \( r_4 \) selected as random numbers between 0 and 1; \( Pi \) is the best destination point position in the \( i \)th dimension, and \(|| \) refers to the absolute value. The remaining parameters \( r_1, r_2, r_3 \) and \( r_4 \) require more elaboration due to their significance for the exploration and exploitation of the SCA. The \( r_1 \) parameter refers to the search circle radius. Through the iteration phase, \( r_1 \) can also be adaptively and dynamically varied using:

\[ r_1 = a^{1-t} \]

where
- \( t \) = current iteration,
- \( T \) = maximum iteration number,
- \( a \) = constant.

Because of the cyclic nature of sine and cosine, the \( r_2 \) parameter determines the updating distance whether the movement is inward (the direction of exploitation when sine and cosine are negative) or outward (exploration direction when sine and cosine are positive), as shown in Figure 1 and Figure 2.

![Figure 1. Effects of Sine and Cosine on search radius.](image)

\( r_3 \) makes a suitable trade-off between emphasizing or deemphasizing the impact of desalination in the distance definition by allocating random weights to the best solution so far acquired. Finally, \( r_4 \) chose equation 1 or 2 depends on a random value between 0 and 1 if \( r_4 \) less then 5.0 take equation 1 else take equation 2.
To clarify, Algorithm 1 provides the general pseudo-code for the SCA.

3.1 Algorithm 1: SCA Pseudo Code
Initialize a set of search agents (solutions)
Initialize the parameter T (maximum number of iterations) and coefficient A
Select the fittest solution Xbest from the set of solutions
Initialize the iteration count t=0 While t< T If \( r_t < 0.5 \) go to
Update each solution vector with the help of equation (1)
Else Update each solution vector with the help of equation (2).
Update A value help with equation (3)
Compute the fitness of each updated solution vector
Update the coefficient A \( T=t+1 \)

4. Results and Discussion
In the area of optimization using meta-heuristics and evolutionary algorithms, several test cases should be used to determine the algorithm's efficiency. Our studies concentrate on two associated objectives: first characterizing the SCA's efficiency by repeating the experiment for choosing the best test ten times, and second benchmarking the SCA and other meta-heuristic approaches. The results show that the SCA introduced performs better than other meta-heuristic algorithms. We have divided our experiments with four groups to achieve these objectives. In Table 1, we take a system configuration \( t=2 \) with 2, 3, 4, 5, 6 V-valued where parameters are varied from 3 to 12. In Table 2, we take a system configuration with \( t=3 \) and Value 3, 4, 5, 6 and 7 where parameters are varied from 4 to 12. In the Table 3, system configuration \( t=4 \) and V varies from 3 to 6 and parameters varies from 5 to 12. All these experiments show SCA with 200 iteration and we select the best one from the 10 experiments. As mentioned in Table 4, we made comparisons with some existing strategies as preliminary evaluation of SCA in terms of the generated test size. The strategies selected for comparison are Jenny [43], PICT [44], tConfig [45], IPOG-D, IPOG [46, 47], DPSO [48, 49], PSTG [50], and CS [15, 51]. We compare our strategy with others by taking parameters (varying from 3 till 12 with 3 values) as a system configuration and we changed the interaction strength (t) from 2 to 11. Table 4 demonstrates the results obtained for the above experiment. The darkened cells show the better results obtained for a specific test configuration. Cells marked NS (not supported) indicate that the tool could not generate the test case for a specific configuration. And cells marked Day means it took a long time. Concerning the size of the generated test suite, Jenny, TConfig, PICT and PSTG did not produce good results with an optimum test suite. Nonetheless \( T=7 \) IPOG, IPOG-D, PSTG and CS were not supported. We note that DPSO and SCA produced optimal results. Where \( t=9 \) DPSO took a long time. Finally, we can note that our SCA strategy compares well against other strategies). As mentioned, that SCA has produced competitive test suites for most strategies and tools.

Table 1. Generated Test Suite Size With \( T=2 \).
### Table 2. Generated Test Suite Size With T=3.

| Configuration | 1st | 2nd | 3rd | 4th | 5th | 6th | 7th | 8th | 9th | 10th | Best | Mean |
|---------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|
| CA(N; t, 3^3) | 9   | 10  | 10  | 9   | 10  | 9   | 10  | 9   | 9   | 9    | 9.7  |
| CA(N; t, 3^4) | 12  | 10  | 10  | 9   | 12  | 10  | 9   | 10  | 9   | 10.2 |
| CA(N; t, 3^5) | 11  | 13  | 13  | 14  | 13  | 14  | 12  | 13  | 14  | 12.8 |
| CA(N; t, 3^6) | 15  | 15  | 14  | 15  | 15  | 14  | 15  | 14  | 14  | 14.7 |
| CA(N; t, 3^7) | 16  | 16  | 16  | 15  | 16  | 15  | 15  | 16  | 14  | 15.4 |
| CA(N; t, 3^8) | 17  | 16  | 16  | 16  | 19  | 15  | 16  | 15  | 16  | 16.3 |
| CA(N; t, 3^9) | 16  | 17  | 17  | 16  | 17  | 17  | 16  | 17  | 16  | 16.7 |
| CA(N; t, 3^10) | 18  | 17  | 17  | 18  | 17  | 18  | 17  | 19  | 18  | 17   |
| CA(N; t, 3^11) | 17  | 18  | 18  | 19  | 18  | 19  | 18  | 19  | 18  | 18.2 |
| CA(N; t, 3^12) | 18  | 17  | 18  | 19  | 18  | 19  | 19  | 19  | 19  | 18.3 |
| CA(N; t, 4^4) | 30  | 30  | 30  | 30  | 30  | 30  | 30  | 30  | 30  | 30   |
| CA(N; t, 5^5) | 48  | 48  | 48  | 46  | 47  | 46  | 47  | 48  | 47  | 46   |
| CA(N; t, 6^10) | 67  | 66  | 66  | 66  | 67  | 66  | 66  | 66  | 66  | 66.2 |
| CA(N; t, 2^7) | 15  | 17  | 18  | 19  | 18  | 19  | 18  | 19  | 18  | 17   |
| CA(N; t, 3^7) | 15  | 15  | 15  | 15  | 15  | 15  | 15  | 15  | 15  | 15.0 |
| CA(N; t, 4^7) | 25  | 26  | 24  | 25  | 28  | 26  | 25  | 26  | 26  | 25.7 |
| CA(N; t, 5^7) | 39  | 41  | 38  | 37  | 40  | 39  | 38  | 39  | 37  | 38.9 |
| CA(N; t, 6^7) | 54  | 56  | 54  | 54  | 54  | 56  | 56  | 57  | 54  | 55.1 |
| CA(N; t, 7^7) | 74  | 74  | 75  | 73  | 74  | 73  | 75  | 73  | 76  | 73.9 |

### Table 3. Generated Test Suite Size With T=4.

| Configuration | 1st | 2nd | 3rd | 4th | 5th | 6th | 7th | 8th | 9th | 10th | Best | Mean |
|---------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|
| CA(N; t, 3^5) | 103 | 100 | 98  | 98  | 101 | 100 | 106 | 97  | 103 | 96   | 96   | 100.7 |
### Table 4. Size performance for CA(N; 4, 3P) where k is varied from 3 to 12.

| T | P | V | Jenny | tConfig | PICT | IPOG-D | IPOG | DPSO | PSTG | CS  | SCA |
|---|---|---|-------|--------|------|--------|------|------|------|-----|-----|
| 2 | 3 | 3 | 9     | 10     | 10   | 15     | 9    | 9    | 9    | 9   | 9   |
| 2 | 4 | 3 | 10    | 10     | 13   | 15     | 9    | 9    | 9    | 9   | 9   |
| 2 | 6 | 3 | 14    | 14     | 13   | 15     | 15   | 11   | 12   | 11  | 12  |
| 4 | 5 | 3 | 16    | 16     | 16   | 15     | 15   | 15   | 15   | 15  | 15  |
| 6 | 7 | 3 | 18    | 18     | 17   | 15     | 17   | 15   | 15   | 16  | 16  |
| 8 | 9 | 3 | 19    | 17     | 18   | 15     | 16   | 17   | 17   | 17  | 17  |
| 10| 11| 3| 17    | 19     | 20   | 18     | 17   | 17   | 17   | 18  | 18  |
| 12| 12| 3| 34    | 32     | 34   | 27     | 32   | 27   | 27   | 28  | 28  |
| 3 | 4 | 3 | 40    | 40     | 40   | 40     | 40   | 40   | 40   | 38  | 38  |
| 6 | 5 | 3 | 51    | 48     | 48   | 45     | 46   | 33   | 45   | 43  | 43  |
| 7 | 7 | 3 | 51    | 51     | 51   | 50     | 55   | 48   | 50   | 48  | 48  |
| 8 | 9 | 3 | 58    | 58     | 59   | 50     | 56   | 52   | 54   | 53  | 52  |
| 10| 11| 3| 62    | 64     | 63   | 71     | 63   | 56   | 58   | 58  | 56  |
| 12| 12| 3| 68    | 77     | 72   | 76     | 73   | 65   | 67   | 67  | 67  |
| 4 | 5 | 3 | 109   | 97     | 100  | 162    | 97   | 81   | 96   | 94  | 96  |
| 6 | 7 | 3 | 140   | 141    | 142  | 162    | 141  | 131  | 133  | 132 | 131 |
| 7 | 9 | 3 | 169   | 166    | 168  | 226    | 167  | 150  | 155  | 154 | 150 |
| 10| 11| 3| 187   | 190    | 189  | 226    | 192  | 171  | 175  | 173 | 171 |
| 9 | 12| 3| 206   | 213    | 211  | 260    | 210  | 187  | 195  | 195 | 191 |
| 10| 13| 3| 221   | 235    | 231  | 278    | 233  | 206  | 210  | 211 | 206 |
| 11| 14| 3| 236   | 258    | 249  | 332    | 251  | 221  | 222  | 229 | 227 |
| 12| 15| 3| 252   | 272    | 269  | 332    | 272  | 237  | 244  | 253 | 246 |
| 6 | 7 | 3 | 1089  | 921    | 1015 | 1201  | 921  | 729  | 977  | 973 | 957 |
| 8 | 8 | 3 | 1466  | 1515   | 1455 | 1763  | 1493 | 1409 | 1402 | 1401 | 1398 |
| 9 | 9 | 3 | 1840  | 1931   | 1818 | 2526  | 1889 | 1682 | 1684 | 1689 | 1698 |
| 10| 10| 3| 1260  | Day    | 2165 | 2834  | 2262 | 1972 | 1980 | 2027 | 1980 |
| 11| 11| 3| 2459  | Day    | 2496 | 3886  | 2607 | 2250 | 2255 | 2298 | 2282 |
| 12| 12| 3| 2757  | Day    | 2815 | 4087  | 3649 | 2512 | 2528 | 2638 | 2537 |
| 7 | 8 | 3 | 3402  | day    | 3143 | Ns    | Ns   | 2250 | Ns   | Ns  | 3020 |
| 9 | 9 | 3 | 4746  | day    | 4618 | Ns    | Ns   | 4451 | Ns   | Ns  | 4414 |
| 10| 10| 3| 6030  | day    | 5884 | Ns    | Ns   | 5443 | Ns   | Ns  | 5443 |
| 11| 11| 3| 7194  | Day    | 7116 | N    | Ns   | 6471 | Ns   | Ns  | 6467 |
| 12| 12| 3| 8296  | Day    | 8314 | Ns    | Ns   | Day  | Ns   | Ns  | 7615 |
| 8 | 9 | 3 | 10572 | day    | 9763 | Ns    | Ns   | 6667 | Ns   | Ns  | 9302 |
| 10| 10| 3| 14999 | Day    | 14599| Ns    | Ns   | 13933| Ns   | Ns  | 13956|
| 11| 11| 3| 19369 | Day    | 18859| Ns    | Ns   | Day  | Ns   | Ns  | 17800|
5. Conclusion and Future Directions
In this paper, we have mentioned an experimental study on meta-heuristic strategy and acceptance mechanism for the generation of combinatorial (t-way tests). The SCA evaluation focuses on the strategy's effectiveness in generating a better size of test suites against current strategies that support uniform.

As shown in table 8 their different sets of experiments were implemented. The main aim here was to show that the strategy was sufficiently competitive compared to other strategies in terms of the generated test suite size. Finally, the results show that in supporting uniform interaction, SCA can compete with other current strategies.

References
[1] Kuhn, D.R. and M.J. Reilly. An investigation of the applicability of design of experiments to software testing. in 27th Annual NASA Goddard/IEEE Software Engineering Workshop.
[2] Kuhn, D.R., D.R. Wallace, and A.M. Gallo, Software fault interactions and implications for software testing. IEEE transactions on software engineering, 2004. 30(6): p. 418-421.
[3] Hariri, F., et al. Comparing Mutation Testing at the Levels of Source Code and Compiler Intermediate Representation 12th IEEE Conference on Software Testing, Validation and Verification (ICST) 2019.
[4] Papadakis, M., et al., Mutation testing advances: an analysis and survey Advances in Computers. 2019 Elsevier. p. 275-378.
[5] Chen, T.Y., S.C. Cheung, and S.M. Yiu, Metamorphic testing: a new approach for generating next test cases. 1998, Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong.
[6] Ma, L., et al., Combinatorial testing for deep learning systems. arXiv preprint arXiv:1806.07723, 2018.
[7] Ghandehari, L.S., et al., A combinatorial testing-based approach to fault localization. IEEE Tran. Software Engineering.
[8] Lee, C.K.H., A review of applications of genetic algorithms in operations management Eng. Applications of Artificial Intelligence 76 1-12.
[9] Jain, N., U. Nangia, and J. Jain, A review of particle swarm optimization. J. Inst. Eng. (India): Series B 2018. 99(4): 407-411.
[10] Zhang, T. and Z.W. Geem, Review of harmony search with respect to algorithm structure. Swarm and Evolutionary Computation 2019 48 31-43.
[11] Dorigo, M. and T. Stützle, Ant colony optimization: overview and recent advances Handbook of metaheuristics. 2019, Springer. p. 311-351.
[12] Streif, M. and M. Leib, Comparison of QAOA with Quantum and Simulated Annealing. arXiv preprint arXiv:1901.01903, 2019.
[13] Abdel-Basset, M. and L.A. Shawky, Flower pollination algorithm: a comprehensive review. Artificial Intelligence Review, 2018: p. 1-25.
[14] Joshi, A., et al., Cuckoo Search Optimization-A Review. Materials Today: Proceedings, 2017. 4(8): p. 7262-7269.
[15] Ahmed, B.S., T.S. Abdulsamad, and M.Y. Potrus, Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the cuckoo search algorithm. Information and Software Technology, 2015. 66: p. 13-29.
[16] Zamli, K.Z., et al., An experimental study of hyper-heuristic selection and acceptance mechanism for combinatorial t-way test suite generation. Information Sciences, 2017. 399: p. 121-153.
[17] Mirjalili, S., SCA: a sine cosine algorithm for solving optimization problems. Knowledge-Based Systems, 2016. 96: p. 120-133.
[18] Ammann, P. and J. Offutt, Introduction to software testing. 2016: Cambridge University Press.
[19] Calvagna, A. and A. Gargantini, IPO-s: incremental generation of combinatorial interaction test data based on symmetries of covering arrays International Conference on Software Testing, Verification, and Validation Workshops 2009.
[20] Yu, L., et al. Acts: A combinatorial test generation tool IEEE Sixth International Conference on Software Testing, Verification and Validation 2013.
[21] Ghandehari, L.S., et al. BEN: A combinatorial testing-based fault localization tool IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 2015.
[22] Yuan, X., M.B. Cohen, and A.M. Memon, GUI interaction testing: Incorporating event context IEEE Tran. Soft. Eng. 2010 37(4): 559-574.
[23] Yuan, X., M. Cohen, and A.M. Memon. Covering array sampling of input event sequences for automated GUI testing Proc. twenty-second IEEE/ACM international conference on Automated software engineering 2007.
[24] Souto, S. and M. d’Amorim Time-space efficient regression testing for configurable systems. Journal of Systems and Software 2018. 137: p. 733-746.
[25] Torres-Jimenez, J. and I. Izquierdo-Marquez Survey of covering arrays 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing 2013.
[26] Lawrence, J., et al., A survey of binary covering arrays elect. j. of combinatorics 2011 18(1): p. 84.
[27] Pradhah, D., et al., Search-based test case implantation for testing untested configurations. Information and Software Technology, 2019. 111: p. 22-36.
[28] Nasser, A.B. and K.Z. Zamli. Self-adaptive Population Size Strategy Based on Flower Pollination Algorithm for T-Way Test Suite Generation. in International Conference of Reliable Information and Communication Technology. 2018. Springer.
[29] Soh, Z.H., et al., Code Coverage of T-way Test Suite Data on Distributed Environment. Int. J. Advance Soft Comp. Appl, 2018. 10(1).
[30] Fujimoto, S., H. Kojima, and T. Tsuchiya, Controlling Occurrence Frequencies of Parameter Values in Pair-Wise Testing. International Journal of Software Engineering and Knowledge Engineering, 2018. 28(07): p. 983-1000.
[31] Koc, U. and C. Yilmaz, Approaches for computing test-case-aware covering arrays. Software Testing, Verification and Reliability, 2018. 28(7): p. e1689.
[32] Drave, I., et al. Model-Based Testing of Software-Based System Functions. in 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). 2018. IEEE.
[33] Kleine, K. and D.E. Simos, An efficient design and implementation of the in-parameter-order algorithm. Mathematics in Computer Science, 2018. 12(1): p. 51-67.
[34] Zamli, K.Z., A.A. Alsewari, and M.I. Younis, T-Way Testing Strategies: Issues, Challenges, and Practices, in Computer Systems and Software Engineering: Concepts, Methodologies, Tools, and Applications. 2018. IGI Global. p. 2011-2024.
[35] Ahmed, B.S., Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing. Engineering Science and Technology, an International Journal, 2016. 19(2): p. 737-753.
[36] Choi, E.-H., et al. Test effectiveness evaluation of prioritized combinatorial testing: A case study. in 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS). 2016. IEEE.
[37] Wang, Z., B. Xu, and C. Nie. Greedy heuristic algorithms to generate variable strength combinatorial test suite. in 2008 The Eighth International Conference on Quality Software. 2008. IEEE.

[38] Ahmed, B.S. and K.Z. Zamli, A variable strength interaction test suites generation strategy using particle swarm optimization. Journal of Systems and Software, 2011. 84(12): p. 2171-2185.

[39] Alsewari, A.A. and K.Z. Zamli. Interaction test data generation using harmony search algorithm. 2011 IEEE Symposium on Industrial Electronics and Applications.

[40] Cohen, M.B., et al. A variable strength interaction testing of components Proceedings 27th Annual International Computer Software and Applications Conference. COMPAC 2003.

[41] Chen, X., et al. Variable strength interaction testing with an ant colony system approach. in 2009 16th Asia-Pacific Software Engineering Conference. 2009. IEEE.

[42] Alsewari, A.R.A. and K.Z. Zamli, Design and implementation of a harmony-search-based variable-strength t-way testing strategy with constraints support. Information and Software Technology, 2012. 54(6): p. 553-568.

[43] Zamli, K.Z., et al., Design and implementation of a t-way test data generation strategy with automated execution tool support. Information Sciences, 2011. 181(9): p. 1741-1758.

[44] Czerwonka, J., Pairwise testing in the real world: Practical extensions to test-case scenarios. Microsoft Corporation, Software Testing Technical Articles.

[45] Williams, A.W., Determination of test configurations for pair-wise interaction coverage Testing of Communicating Systems 2000 Springer p. 59-74.

[46] Lei, Y., et al., IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing Software Testing, Verification and Reliability, 2008. 18(3): p. 125-148.

[47] Lei, Y., et al. IPOG: A general strategy for t-way software testing 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07).

[48] Omran, M., A.P. Engelbrecht, and A. Salman, Particle swarm optimization method for image clustering International Journal of Pattern Recognition and Artificial Intelligence 2005 19(03): p. 297-321.

[49] Wu, H., et al., A discrete particle swarm optimization for covering array generation. IEEE Transactions on Evolutionary Computation 2014. 19(4): p. 575-591.

[50] Ahmed, B.S. and K.Z. Zamli. PSTG: a t-way strategy adopting particle swarm optimization Fourth Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation 2010

[51] Li, X. and M. Yin, Modified cuckoo search algorithm with self adaptive parameter method Information Sciences 2015. 298: p. 80-97.