ABC Algorithm for Combinatorial Testing Problem

AbdulRahman A. Alsewari[1,2], Amaar K. Alazzawi[1], Taha H. Rassem[1], Muhammad N. Kabir[1], Ameen A. Ba Homaid[1], Yazan A. Alsariera[1], Nasser M. Tairan[3], and Kamal Z. Zamli[1]
Software Engineering Research Group,
[1]Faculty of Computer Systems and Software Engineering,
[2]IBM Centre of Excellence, Universiti Malaysia Pahang, Pahang, Malaysia.
[3]College of Computer Science, King Khaled University.
alsewari@ump.edu.my

***Abstract*—Computer software is in high demand everywhere in the world. The high dependence on software makes software requirements more complicated. As a result, software testing tasks get costlier and challenging due to a large number of test cases, coupled with the vast number of the system requirements. This challenge presents the need for reduction of the system redundant test cases. A combinatorial testing approach gives an intended result from the optimization of the system test cases. Hence, this study implements a combinatorial testing strategy called Artificial Bee Colony Test Generation (ABC-TG) that helps to get rid of some of the current combinatorial testing strategies. Results obtained from the ABC-TG were benchmarked with the results obtained from existing strategies in order to determine the efficiency of the ABC-TG. Finally, ABC-TG shows the efficiency and effectiveness in terms of generating optimum test cases size of some of the case studies and a comparable result with the existing combinatorial testing strategies.**

***Index Terms*—Computational Intelligence; Combinatorial Optimization Problem; Software Testing; Test Data Generation.**

I. INTRODUCTION

Software systems continue to develop progressively in complexity and size in this era. Software has become gradually ubiquitous in tools and methods used for science, engineering, medicine and human interactions. A software fault is a mistake in the programmed code which, when faced may be the reason for the software's failure. The software behaves in an unexpected way when it encounters faults in it. Different methods are implemented to avoid, notice and rectify the errors throughout the software design life cycle phases. Thus, software testing is a fundamental activity in securing the quality assurance of most software products [1-3]. It is the utmost key in guaranteeing a reliable software product. In software development life cycle, software testing acts as an integral and tedious activity [4].

The presence of faults in a software system can result into unprecedented cost or even life losing [4]. Software testing takes a vital part in inspecting detects via probable test data to make sure it's quality. Most of the software systems of nowadays are produced using components. Most times, the system bugs or errors are as a result of the unexpected fusion between the components used [5-7]. For instance, if the testing of Microsoft's words displays tab is considered in the dialog as seen in Figure 1.

It has seventeen feasible options (parameters P=17) that can take two probable values (V=2) 2^{17}= 131,072 are to be analyzed. These are virtually ineffective. Analyzing a test case requires five minutes, and it will take a whole 15 months to examine only the display tab completely which is not probable practically according to the testing standards.

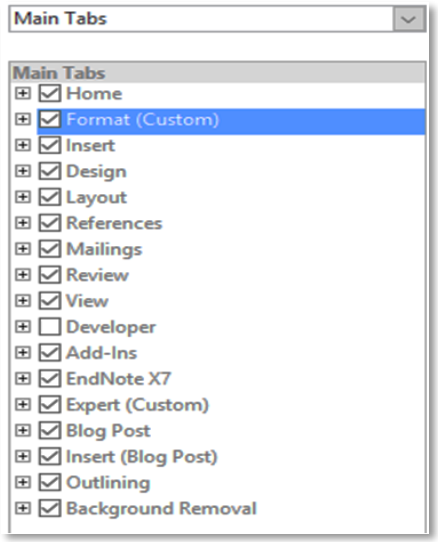

Figure. 1: Microsoft Word, Options, Customize Ribbon

Exhaustive testing is impossible. Thus, there are a lot of strategies have been designed and developed to minimize the test cases based on optimization algorithms Genetic Algorithm (GA) [8], Simulated Annulling (SA) [5, 9], and Harmony Search Strategy (HSS) [1, 2]. The first approach that used to minimize the test cases is the Pairwise Testing approach. Pairwise testing approach is that used to generate test cases based on all possible pairs of system's input values. Pairwise testing is the basic level of combinatorial testing approach. Combinatorial testing approach is the test suites generator that covers all combinations of the system parameters based on the combination degree. Thus, it is much smaller than exhaustive ones yet still very effective in finding defects [10, 11]. However, one of the main complications of Combinatorial is finding a minimal test suite.

To address this issue, many algorithms have been implemented such as: Automatic Efficient Test Generator (AETG) [10], GA [8], In Parameter Order (IPO) and its family (IPOG-D) [12, 13], Test Configuration (TConfig) [14], Pairwise Independent Combinatorial Testing (PICT)[15], Classification-Tree Editor eXtended Logics (CTE-XL)[16], Jenny [17], ITCH[18], Test Vactor Generator (TVG) [19], SA [5, 9], and HSS [1, 2]. It is observed that most of them are not efficient and the existing strategies are based on the optimization algorithms. The current strategies cannot

achieve the optimal balance between exploration and exploitation. The aims of this paper is to present, design and implement a new combinatorial testing strategy based on Artificial Bee Colony, called Artificial Bee Colony Test Generation (ABC-TG) strategy. Then benchmark the ABC-TG's results with the existing combinatorial testing strategies' results to evaluate the efficiency of the ABC-TG strategy.

II. TEST CASES GENERATION ALGORITHM BASED ON ABC-TG STRATEGY

ABC-TG strategy has been applied to solve numerous test cases optimization problems. The position of the food provenance discovered, represents a possible test case of the optimization problem, and the quality (fitness function) of the related test case, corresponds with a nectar amount (represent the combination pairs).

The ABC-TG has three sets of bees; each of them is working to accomplish a certain task. These sets are employed, onlookers, and scouts' bees. In making the decision of selecting a food provenance, a set of bees must be waiting on the dance area, this are called onlooker. The other group which go to visit a food provenance are dubbed employed bee. The last set of bees are the scout bee, they execute random work of discovering new province of food. The discovery of a food provenance represents a possible test case to the optimization problem, and the nectar amount of a food provenance corresponds to the quality [11].

The first part of the algorithm consists of a few numbers of employed artificial bees, while the second part of the algorithm has the onlooker bees. In each part of the algorithm, the employed bees and the onlooker bees will represent the test cases in the population. Pass the ABC-TG strategy in four phases respectively; initialization, employed, onlooker and scout bee's phases (see Figure 2).

1: Generate the initial population test cases (x_i) using Eq(1) , $i= 1...SN$
2: Evaluate the x_i fitness (f_i) of the population
3: Set cycle to 1 , and MCN=number
4: Repeat
5: for each employ bee {
  Produce new solution v_i using Eq(2)
  Evaluate v_i fitness (f_i)
  Select the best test case }
6: Calculate the probability (p_i) for test case (x_i) using Eq(3)
7: for each on looker bee{
  Select a test case x_i based on p_i
  Produce new solution v_i using Eq(2)
  Evaluate v_i fitness (f_i)
  Select the best test case }
8: If there is an abandoned test case for the scout
  Then replace it with a new test case produced randomly using Eq(1)
9: Memorize the best test case
10: cycle=cycle+1;
11 until cycle=MCN

Figure 2.ABC-TG Pseudo-Code

A. Initialization Phase

Initially, the ABC-TG algorithm starts producing randomly distributed population of Solutions size (SN) of test cases (food provenance positions). Where SN shows the size of an onlooker or employed bees [20]. Assuming *D* is the optimization parameter number (System configuration parameters), then every single solution test case (x_i) (i =1, 2... SN) basically will exist as a *D*-dimensional vector. By using the Equation (1) produces all the initial test cases for employed bees.

$$x_{ij} = x_{min,j} + \text{rand}(0,1)(x_{max,j} - x_{min,j}) \quad (1)$$

where the value of x_{min} and x_{max} are sequentially upper & lower limits for the test case variable x_i in dimension j ($j=1, 2... D$), and *rand* is a random digit number scaling factor which is between the number [0, 1]. *D*-dimensional test cases (food provenance positions) created during the initialization stage (C=0) are subject to the cycles of Iterative (C=1, 2…, MCN), till a termination condition is satisfied and are implemented locally and also as a global probabilistic selection/search in a one cycle ABC-TG. Each cycle depicts a total number of tasks made by the different types of bee. Thus, these whole methods are principally independent which can be explicated in a separate form as follows, to have a better understanding of the ABC-TG methodology.

B. Employed Bee Phase

Firstly, the phase of employed bees where creates a new candidate solutions (test cases) by evaluating the capability of the test cases and interchanges the data with the onlooker bees' stage. And the employed bee creates (food position) a candidate test case by the removal of the former (x_{ij}) test case solution in its memory, by utilizing Equation (2) test case only is updated [21].

$$v_{ij}= x_{i,j} + rand[-1,1](x_{ij}- x_{kj}) \quad (2)$$

Here $j\in\{1,2,...,D\}$ and $k\in\{1,2,..., SN\}$ ($k\neq i$) are randomly-selected indexes, & *rand* exists as a random number of [-1, 1], which works as a scaling factor. It is clear that the optimum test case is reached in the search area, this gets reduced because of the disorder in the solution. It evaluates the fitness of the new solution by the employed bee, and it updated the fitness values that is found, and replaces with the new test case instead of the former one in the employed bee's memory (a greedy-selection).

C. Onlooker Bee Phase

In ABC-TG algorithm, the principal task of every single onlooker bee is to indicate a food provenance (test case value) according to probability value, and depending on the fitness value related to the food provenance *Pi*. This can be calculated using Equation (3).

$$\text{Pi}= \frac{fitness_i}{\sum_{n=1}^{sn} fitness_n} \quad (3)$$

Here, fit represents the value of fitness of a certain test case. The probabilistic selection is implemented by making a comparison of Pi against a randomly selected number which is between [0, 1]. The selection is approved if the created random value is equal or less than Pi, and if otherwise, it will be rejected. Thus, the assignment of an onlooker bee to that particular test case will be approved if the conforming probabilistic selection is sanctioned. When evaluating the new solution fitness, the new food provenance (test case) will

be selected by an onlooker bee in the area of the former one which is in her memory, utilizing Equation 2. In case the new test case has a fitness value that is better, then an onlooker bee will make an update on the new test case that exists in her memory and let go of the old one. This is related to the employed bee case.

D. Scout Bee Phase

In this phase, the scout bees work randomly to discover all the search spaces to be able to get a new test case (enhanced) to the problem of the global optimization. On the other hand, unlike the onlooker bees and employed bees (that have a limit to produce a trial test case around the former test case), scout bees are indefinite in this sense. They adopt their samples from a wide range of *D*-dimensional vectors; so far remains in the search space limits. Otherwise, cannot improve the solution (test case) after a determined number of cycles. Formerly, if it is non-global, then it will abandon this test case and the employed bee will be employed to that particular situation which will then be converted to a scout bee with principally scout-type functionality. The process will continue until the exit criteria has been meet

III. BENCHMARKING

To evaluate the ABC-TG strategy, there will be several experiments collected from the publications [1, 2, 5, 9, 12, 13]. ABC-TG strategy has initialized its parameters such as: the number of the improvisation = 80, number of bees = 50, and the value of limit = 100 as suggested by the researchers [21]. The results obtained by ABC-TG strategy based on 20 times of running the experiments in the environment that is composed of PC with Windows 10 Pro 64-bit, Intel Core i7 3.40 GHz, 4.00 GB of RAM. The time of implementation is determined in seconds. The ABC-TG strategies are implemented in Java (JDK 1.8.0.31). The results are presented in Tables, Table 2 and Table 3 compared with the best results obtained by other existing strategies. The darkened cells with bold numbers represent the best results obtained for the test configuration. The results for some strategies are not available through the literature (publications), these cells are marked by NP (not published). Some strategies do not support certain interaction strength, these cells are marked by NS (not supported). The configuration systems used in this evaluation details as: Covering Array (CA), number of systems parameters (P), the values for each parameter represent by (V), and N is the minimum size of test list. For example: CA(16, 2, 4^5). The minimum test list that can be produced N=16, interaction testing degree t=2, system parameters P=5, each parameter has value V=4.

To clarify the performance in terms of the support interaction strength (i.e. $2 \leq t \leq 6$), the following experiments adopted three configurations system from the published results in the works by [1, 2, 5, 9, 12, 13].

In cases with high interaction strength within a configuration system CA (N; t, 3^7) as shown in Table 1, ABC-TG mostly provides an optimal result. However, the ABC-TG does not generate the most optimal results in all cases, but it generates satisfactory results. TConfig, obtained best result when interaction strength is 6, ITCH obtained only one optimum result when the interaction strength is 3.

Table 1.
CA (N; t, 37), t is variable from 2 to 6.

T	Jenny	TConfig	ITCH	PICT	TVG	CTE-XL	IPOG-D	IPOG	ABC-TG			
									Best	B. time	A. size	A. time
2	16	15	15	16	15	16	18	17	15	19.3	15.5	19.5
3	51	55	45	51	55	54	63	57	49	266.2	50.8	270.6
4	169	166	216	168	167	NS	NP	185	158	1918.2	161.3	1930.7
5	458	477	NS	452	464	NS	735	608	443	5741.0	451.3	5761.2
6	1087	921	NS	1015	1016	NS	1548	1281	945	4950.1	977.5	5119.06

Constrain wise, IPOG-D does not support for only one configuration CA (N; 4, 3^7). However, Jenny, PICT, IPOG-D and IPOG commonly produce the worst results overall. On the other hand, when interaction strength is equal to 2, ABC-TG generates the most optimal only once.

As shown in Table 2 with a configuration system CA (N; 3, 3^P), ABC-TG generates the most optimal minimum result. Comparing with the other strategies for only one configuration is CA (N; 3, 3^6). While ITCH, and ABC-TG usually produce the near minimum and comparable results of the final test cases. However, Jenny, Tconfig, PICT, TVG, CTE-XL, IPOG-D and IPOG commonly produce the worst results overall. On the other hand, ABC-TG generated comparable results for two cases namely CA (N; 3, 3^9) CA (N; 3, 3^{10}), Contrariwise other strategies that produce the worst results.

Table 2.
CA (N; 3, 3P), P is variable from 4 to 10.

P	Jenny	TConfig	ITCH	PICT	TVG	CTE-XL	IPOG-D	IPOG	ABC-TG			
									B	b.time	A.size	A.time
4	34	32	27	34	34	34	27	39	33	6.82	34.5	7.16
5	40	40	45	43	41	43	49	43	40	28.33	41.9	28.9
6	51	48	45	48	49	52	49	53	43	90.8	46.8	94.9
7	51	55	45	51	55	54	63	57	50	264.2	52.0	268.2
8	58	58	45	59	60	63	63	63	54	654.2	55.8	663.2
9	62	64	75	63	64	66	71	65	58	1452.9	59.8	1470.1
10	65	68	75	65	68	71	71	68	62	3022.3	63.6	3041.5

Regarding to Table 3, ABC-TG generates the minimum test cases size and the satisfactory results for two configuration systems namely; CA (N; 3, 2^7) and CA (N; 3, 6^7) and outperforms comparing with other strategies. As well as, TConfig, ABC-TG, IPOG-D and ITCH generate the near optimal results in some cases and the competitive results in

the others. Although Jenny, PICT, TVG and CTE-XL does not generate any most minimum result. But usually produces comparable results, while IPOG-D generates the most optimal results only once, and the acceptable results for other strategies.

Table 3.
CA (N; 3, V7), V is variable from 2 to 6.

V	Jenny	TConfig	ITCH	PICT	TVG	CTE-XL	IPOG-D	IPOG	ABC-TG			
									B	B.time	A.size	A. time
2	14	16	13	15	15	15	14	19	12	36.4	14.3	39.6
3	51	55	45	51	55	54	63	57	49	264.07	52	268.7
4	124	112	112	124	134	136	112	208	116	1218.6	120.9	1230.4
5	236	239	225	241	260	267	292	275	228	4315.3	231.4	4360.1
6	400	423	1177	413	464	467	532	455	391	12522.6	394	12588.2

IV. CONCLUSION

This paper proposes a new combinatorial testing strategy called ABC-TG strategies, based on the ABC algorithm which generate test list. The main motivation of the ABC-TG is to reduce the size of final test list. According to different sets of experiments that have been conducted, ABC-TG has shown performance and efficiency in term of generating a near optimal final test list size. As part of our future work, we are planning to enhance the ABC-TG, introduce a variable strength and seeding into the current implementation.

ACKNOWLEDGMENT

This research is funded by UMP RDU150369: A new Hybrid Variable Interaction Strength Test Data Generation Strategy Based on Harmony Search Algorithm and Cuckoo Search Algorithm, UMP RDU1603119 Grant: Modified Greedy Algorithm Strategy for Combinatorial Testing Problem with Constraints Supports. Also it is partially funded by FRGS RDU160102: A New Global Optimization Algorithm based on Stochastic Approach to Minimize Software Testing Redundancy

REFERENCES

[1] A. A. Alsewari and K. Z. Zamli, "Interaction test data generation using harmony search algorithm," in *Proceeding of IEEE Symposium on Industrial Electronics & Applications*, Langkawi, Malaysia, 2011, pp. 559-564.

[2] A. R. A. Alsewari 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*, vol. 54, no. 6, pp. 553-568, 2012.

[3] A. A. Ahmed and C. Xue Li, "Analyzing Data Remnant Remains on User Devices to Determine Probative Artifacts in Cloud Environment," *Journal of Forensic Sciences*, 2017.

[4] B. Hambling, P. Morgan, A. Samaroo, and P. Williams, *Software Testing: An ISTQB-ISEB Foundation Guide*: BCS, The Chartered Institute, 2010.

[5] M. B. Cohen, M. B. Dwyer, and J. Shi, "Interaction testing of highly-configurable systems in the presence of constraints," in *the 2007 international symposium on Software testing and analysis*, 2007, pp. 129-139.

[6] A. A. B. Homaid and A. A. Alsewari, "A variable combinatorial test suite strategy based on modified greedy algorithm," in *Software Engineering and Computer Systems (ICSECS), 2015 4th International Conference on*, 2015, pp. 154-159.

[7] K. Z. Zamli, A. R. Alsewari, and B. Al-Kazemi, "Comparative benchmarking of constraints t-way test generation strategy based on late acceptance hill climbing algorithm," *International Journal of Software Engineering & Computer Sciences (IJSECS)*, vol. 1, pp. 14-26, 2015.

[8] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using artificial life techniques to generate test cases for combinatorial testing," in *the 28th Annual International on Computer Software and Applications Conference, 2004. COMPSAC 2004.* , 2004, pp. 72-77.

[9] J. Stardom, *Metaheuristics and the Search for Covering and Packing Arrays*. Simon Fraser University, 2001.

[10] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: an approach to testing based on combinatorial design," *Software Engineering, IEEE Transactions on*, vol. 23, no. 7, pp. 437-444, 1997.

[11] D. V. Reddy and A. R. M. Reddy, "An approach for fault detection in software testing through optimized test case prioritization," *International Journal of Applied Engineering Research*, vol. 11, no. 1, pp. 57-63, 2016.

[12] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing," *Software Testing, Verification and Reliability*, vol. 18, no. 3, pp. 125-148, 2008.

[13] Y. Lei and K.-C. Tai, "In-parameter-order: a test generation strategy for pairwise testing," in *the Third IEEE International on High-Assurance Systems Engineering Symposium, 1998*, 1998, pp. 254-261.

[14] A. Williams, J. Lo, and A. Lareau, "TConfig," ed, 2010.

[15] J. Czerwonka, D. Butt, and C. Gens, "Pairwise testing in real word: practical extensions to test case generators," in *Proc. of the 24th pacific northwest software quality conf. 2006*, 2006.

[16] E. Lehmann and J. Wegener, "Test case design by means of the CTE XL," in *Proceedings of the 8th European International Conference on Software Testing, Analysis & Review (EuroSTAR 2000), Kopenhagen, Denmark*, 2000.

[17] Jenkins, "Test Tool," 2003. Available at http://www.burtleburtle.net/bob/math/jenny.html.

[18] A. Hartman, T. Klinger, and L. Raskin, "IBM intelligent test case handler," *Discrete Mathematics*, vol. 284, pp. 149-156, 2010.

[19] P. J. Schroeder, E. Kim, J. Arshem, and P. Bolaki, "Combining behavior and data modeling in automated test case generation," in *the Third International Conference on Quality Software, 2003.*, 2003, pp. 247-254.

[20] B. Nozohour-leilabady and B. Fazelabdolabadi, "On the Application of Artificial Bee Colony (ABC) Algorithm for Optimization of Well Placements in Fractured Reservoirs; Efficiency Comparison with the Particle Swarm Optimization (PSO) methodology," *Petroleum*, 2015.

[21] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied mathematics and computation*, vol. 214, no. 1, pp. 108-132, 2009.