ABCVS: An Artificial Bee Colony for Generating Variable T-Way Test Sets

Ammar K Alazzawi[1], Helmi Md Rais[2], Shuib Basri[3]
Department of Computer and Information Sciences, Faculty of Science and Information Technology
Universiti Teknologi Petronas, Bandar Seri Iskandar 32610, Perak, Malaysia

***Abstract*—To achieve acceptable quality and performance of any software product, it is crucial to assess various software components in the application. There exist various software-testing techniques such as combinatorial testing and covering array. However, problems such as *t*-way combinatorial explosion is still challenging in any combinatorial testing strategy, as it takes into consideration the entire combinations of input variables. Therefore, to overcome this problem, several optimizations and metaheuristic strategies have been suggested. One of the most effective optimization algorithms based techniques is the Artificial Bee Colony (ABC) algorithm. This paper presents *t*-way generation strategy for both a uniform and variable strength test suite by applying the ABC strategy (ABCVS) to reduce the size of the test suite and to subsequently enhance the test suite generation interaction. To assess both the effectiveness and performance of the presented ABCVS, several experiments were conducted applying various sets of benchmarks. The results revealed that the proposed ABCVS outweigh the existing based strategies and demonstrated wider interaction between components as opposed to AI-search based and computational based strategies. The results also revealed higher prospect of ABCVS in the aspect of its effectiveness and performance as observed in the majority of case studies.**

Keywords*—*T-way testing; variable-strength interaction; combinatorial testing; covering array; test suite generation; artificial bee colony algorithm

I. Introduction

For investigate acceptable quality and performance for any software product, it is important to assess the various software components in the application. Software testing is afforded with many techniques and tools, divided into two major categories; Black box and White box testing. The inner components of the system under test (SUT) in performing white-box testing, are considered during the generation of the test case, while the input variables and how they interact as a significant function in black-box testing, occurs during test suite production [1]. Due to the complexity of the most software systems, therefore, all-inclusive testing taking the entire configurations and interactions into consideration is not feasible due to computational constraints as well as the need for sampling strategies [2]. From several of the techniques available for black box software, both combinatorial testing (CT) in addition to covering array (CA) are suitable approaches used for testing purposes. In fact, some studies of *t*-way interaction testing have indicated this form of testing to be effective in identifying nearly all of the flaws in a typical software system [3]. Notably, various uniform and variable strength *t*-way techniques are documented throughout the literature in this domain.

However, combinatorial explosion in a combinatorial testing strategy such as a *t*-way is a problem. Moreover, while it is not feasible to generate the entire combination of variables in this case, it is necessary that a CA of minimum size be generated. Generally, the minimum size of a CA is an unspecified priori. Therefore, it is advisable to generate as much CA as possible with several test cases within an acceptable amount of time. Various strategies have been found in the literature to produce the test suite size with uniform and variable strength interaction. Accordingly, to assess such a strategy, three features are required, namely; 1) the array size of the test suites that denote the effectiveness, 2) speed of the strategy, which refers to performance, and 3) portability of the strategy to support adequate high interaction strength. Different approaches have been adopted using numerous strategies, such as; Artificial Intelligence (AI) based and Pure Computational-based approaches [4]. These strategies are based on pure computational-based approaches which are characterized by their high performance but having worst efficiency, for instance; In-Parameter-Order-General (IPOG-D) [5] and Intelligent Test Case Handler (ITCH) [6]. While ITCH generates test suites of small array sizes, the IPOG D strategy, on the other hand, has a fast-paced approach. Previous studies have suggested that when a small interaction strength is considered (i.e. $t \leq 3$), it can cover most flaws in a typical software system [7-12]. Subsequent research studies although, have proposed the need to support the higher strength interaction, particularly for complex systems [2, 4, 13-15].

The appropriate solution to address the problems associated with the computation of the minimal test suite is by employing a search-based software engineering (SBSE) technique [16-19]. By drawing inspiration from the SBSE technique, various strategies have already been suggested towards the problem of uniform and variable strength interaction (e.g. Simulated Annealing (SA) [9-11, 20, 21], Genetic Algorithm (GA) [12, 21-27], Practice Swarm Optimization (PSO) [7, 28-30], Ant Colony Algorithm (ACA) [8, 12], Harmony Search Algorithm (HS) [4, 31, 32], Cuckoo Search (CS) [3, 33, 34], and the Bat Algorithm (BA) [35-39]). Even though relevant strategies have been found to be useful based on AI, they are noted in the literature as being slow due to costly computations [3, 4, 7]. Another challenge encountered for some AI-based strategies that are considered to be slow is support for small interaction strengths (i.e. $t \leq 4$) for generating test suites. Therefore, in this paper, the ABC algorithm is proposed as an efficient and acceptable strategy, known as the Artificial Bee Colony Strategy (ABCVS) continuous to our previous research [40, 41] for uniform and variable strength interaction *t*-way minimal test suite production. It is anticipated that ABCVS will address this problem. In fact, experimental results have revealed that the proposed ABCVS is able to support higher interaction strengths up to $t = 6$ compared to other Artificial Intelligence (AI)-based strategies. Furthermore, ABCVS it can compete against these other strategies in the majority of the case studies examined in the literature regarding efficiency (test suite generation) and performance (speed) and against other AI-based strategies having higher interaction strengths.

The remainder of this paper is structured accordingly. Section II presents the background to this study, (i.e. CA and MCA concepts) which is followed by Section III which surveys 'state of the art' testing strategies in this area. Section IV provides an overview of an ABC which is then followed by Section V describing the proposed strategy, consisting of two parts: (1) construction of the covering matrix, and (2) the proposed ABCVS. Section VI illustrates the tuning of the ABCVS parameters. Section VII evaluates the ABCVS by conducting several benchmark experiments in terms of effciency and performance alongside with statistical analysis evaluation by using Wilcoxon signed-rank test in Section VIII. Section IX discusses the advantages, limitations and threats to the validity of the approach, which followed by Section X providing overall conclusions to the study and presenting recommendations for future work.

II. Covering Array with Problem Definition Model

The strategy of a *t*-way testing is one of the most minimization criteria used regarding the number of test cases list. The *t*-way testing is a combinatorial approach, and "*t*" is indicates the interaction strength [12], and one of the input parameters. For example, assume that the (SUT) behavior is represented by user input then the external and internal events or the modes of the system parameters are controlled by separate parameters (P). In this case, each separate value Vi has a parameter, Xi. These Vi are selected from a fixed set of values, consisting of value members. Each p-tuple…(X1, X2, …,Xp) shows a test case where $Xi \in Vi$ and $1 \leq i \leq P$ [29].

Using a practical software example, let us consider a web configurable software system. This system is consists of five-components, namely; a device, a processor, the operating system (OS), browser, and a display. Fig. 1 illustrates their relationships. This system has employed as a simple illustration of the main idea in *t*-way testing regarding variable strength, and uniform strength. Each component of this system is known as a separate parameter, and each parameter has one or more values. Table I displays the parameters of the system and their values. In this example, the number of parameters is five, consisting of 3-parameters with 2-values and 2-parameters with 2-values.

To produce different software settings of the system, the software should be tested to identify any existing defects. In this case, a test case can be used to determine particular settings of the software. The total number of test cases are called the test suite, where the test suite must identify existing defects. For instance, assuming device 1 (Phone) is not able to use OS 2 (Windows). Therefore, the system will fail because the operating system (OS) is Windows based and the device parameter is the phone. Therefore, to detect faults, using 2-way schema (Phone,-, Windows,-,-) at least one occurrence time must be covered by the generated test suite.

To obtain 'perfect' software that performs with no defects, it is important to test all software components to cover all possible parameters. In this example, 72 test cases (i.e. $2 \times 2 \times 3 \times 3 \times 2$) are required to cover all cases. Nowadays, software-testing costs are rising due to the evolution and complexity of the software and an increasing the number of parameters and software levels. Since the failure of a limited number of parameters is mainly caused by the interaction of the parameters [12], it is not efficient to produce a comprehensive test case. Therefore, to solve this problem, a particular technique should be used such as a *t*-way testing approach as an alternative to a more comprehensive test, where $2 \leq t \leq p$. In addition, it is worth noting that the "*t*" value denotes that the final test suite consists of all possible *t*-way combinations as an alternative to the p-way. However, it cannot identify the interactions that are responsible for the software failure caused if the "*t*" value is low.

All possible interaction combinations of the parameters are covered by the combinatorial test, called the *t*-way CA for the test that generates the test suite. In addition, the "*t*" value defines the covering depth as well as the strength, and it is considered an important factor that must be determined by the tester. Overall, mathematically both the *t*-way approach and CA have a direct relationship. Therefore, to define the *t*-way approach, the following concept is used where CA is an array size N × p, where N contains test cases, consisting of two features:

Each column i contains some members of set Vi, |Vi| = Vi. The rows of each sub-array of size N × *t*, cover all combinations $|Vk_1| \times |Vk_2| \times \ldots \times |Vk_t|$ of t at least once [29]; and if $v_1 = v_2 = \ldots == v_p = v$, then all possible cases are obtained as given in Eq. (1)

$$\text{FullCoverage} = \binom{P}{t} \text{ x vt} \tag{1}$$

The CA is denoted by CA (N; *t*, p, v_1, v_2, …..,vp), and if $v_1 = v_2 = \ldots = v^P = v$, then it is denoted by CA (N; *t*, p, v) or CA (N; *t*, v^P) [29]. In some research references, if vi is different, it is called a Mixed Covering Array (MCA) [38, 42-44].

In the earlier section, it was mentioned that the possible number of test cases of a web configurable software system was 72. In this case, 2-way (pairwise interaction) is used where the total number of test cases has been reduced to nine as shown in Table II, which displays all possible combinations for available parameters. For example, both the parameters, device and the OS, have six (i.e. 2 × 3) different 2-way cases including (Pc, -, Android , -, -), (Pc, -, IOS, -, -), (Phone, -, Android , -, -), (Phone, -, IOS, -, -), (Phone, -, Windows, -, -) and (Pc, -, Windows, -, -), (see Table II).

The interactions between the parameters are important given that most software is unstable, due to some of these interactions having less impact. Some others interactions are more likely related to system failure. In this case, CA along with the variable strength is used effectively to determine the various interactions. Various covering strengths exist in this structure between the various sets of parameters. The VS Covering Array (VSCA) is denoted by VSCA (N; *t*, *p*, *v*, {C}) where {C}, is consists of one or more CA (*t'*, *p'*, *v*) and p' is a subset of p. A web configurable software system example is next shown to understand VSCA. In the web configurable software assumes that all parameter combinations require 2-way and a device, processor and OS that requires 3-way interaction. In this case, the configuration system is indicated as VSCA (N; 2, $2^3 3^2$, CA (3, $2^2 3^1$)). Table III displays the complete test suite to cover this configuration. Three more test cases as shown in Table III, are added to Table II in order to construct VSCA (12; 2, $2^3 3^2$, CA (3, $2^2 3^1$)). For this reason, using VSCA not only covers all 2-way interactions of complete parameters, but also covers all 3-way interactions of the specified parameters.

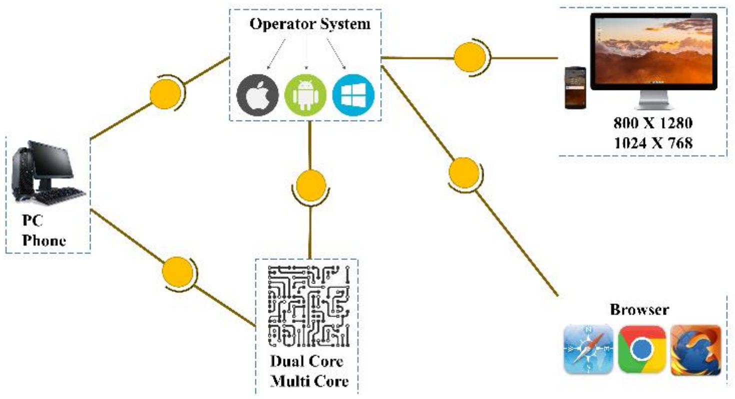

Fig. 1. A Web Configurable Software System..

TABLE I. A WEB CONFIGURABLE SOFTWARE SYSTEM

Parameters				
Device	**Processor**	**OS**	**Browser**	**Display**
PC (0)	Dual core (0)	Android (0)	Firefox (0)	720 x 1280 (0)
Phone (1)	Multi core (1)	IOS (1)	Chrome (1)	1024 x 768 (1)
		Windows (2)	Safari (2)	

TABLE II. TEST SUITE FOR CA (N; 2, $2^3 3^2$)

NO	Device	Processor	OS	Browser	Display
1	PC (0)	Dual core (0)	IOS (1)	Firefox (0)	720 x 1280 (0)
2	Phone (1)	Multi core (1)	Android (0)	Firefox (0)	1024 x 768 (1)
3	Phone (1)	Multi core (1)	Windows (2)	Chrome (1)	720 x 1280 (0)
4	PC (0)	Dual core (0)	Windows (2)	Safari (2)	1024 x 768 (1)
5	PC (0)	Dual core (0)	Android (0)	Chrome (1)	1024 x 768 (1)
6	Phone (1)	Multi core (1)	IOS (1)	Safari (2)	1024 x 768 (1)
7	PC (0)	Multi core (1)	Android (0)	Safari (2)	720 x 1280 (0)
8	Phone (1)	Dual core (0)	Windows (2)	Firefox (0)	1024 x 768 (1)
9	Phone (1)	Dual core (0)	IOS (1)	Chrome (1)	1024 x 768 (1)

TABLE III. TEST SUITE FOR VSCA (N; 2, $2^3 3^2$, CA (N; 3, $2^2 3^1$))

NO	Device	Processor	OS	Browser	Display
1	PC (0)	Dual core (0)	IOS (1)	Firefox (0)	800 x 1280 (0)
2	Phone (1)	Multi core (1)	Android (0)	Firefox (0)	1024 x 768 (1)
3	Phone (1)	Multi core (1)	Windows (2)	Chrome (1)	800 x 1280 (0)
4	PC (0)	Dual core (0)	Windows (2)	Safari (2)	1024 x 768 (1)
5	PC (0)	Dual core (0)	Android (0)	Chrome (1)	1024 x 768 (1)
6	Phone (1)	Multi core (1)	IOS (1)	Safari (2)	1024 x 768 (1)
7	PC (0)	Multi core (1)	Android (0)	Safari (2)	800 x 1280 (0)
8	Phone (1)	Dual core (0)	Windows (2)	Firefox (0)	1024 x 768 (1)
9	Phone (1)	Dual core (0)	IOS (1)	Chrome (1)	1024 x 768 (1)
10	PC (0)	Dual core (0)	Android (0)	Firefox (0)	1024 x 768 (1)
11	PC (0)	Multi core (1)	IOS (1)	Chrome (1)	800 x 1280 (0)
12	Phone (1)	Dual core (0)	Windows (2)	Firefox (0)	800 x 1280 (0)

III. Related Wor k

Throughout Over the last few decades, there have been many studies in the literature that grouped into four main sets to solve the CA problem [45]: (A) mathematic methods, (B) greedy algorithms, (C) heuristic search algorithms, and (D) random methods.

A. *Mathematic Methods*

The mathematician researcher uses mathematic methods derived from the extensions of mathematical functions like the Orthogonal Array (OA). To generate CA for optimal mathematics, it is only available for specific configurations (i.e. the parameters and values are required to be uniform) [12]. Notably, there are several strategies designed based on this approach such as TConfig [46] and Combinatorial Test Service (CTS) [47] strategies for the generalisation of an OA. However, a major problem exists for the OA (i.e. as the solution is only available for small configurations because of the restriction).

B. *Greedy Algorithms*

Most greedy algorithm (GA) solutions initially begin to generate all possible combinations (i.e. solutions) based on the input variables; then, generate one test at a time until all uncovered combinations are covered. A greedy algorithm is different in this case compared to the other strategies for generating test cases. In order to produce a test case using greedy algorithms [3], there are two main approaches; (1) One-test-at-a-time (one-line-at-time) and (2) One parameter-at-a-time.

In each iteration, the one-test-at-a-time (OTAT) approach adds a line (test case) to the test suite. The strategies that adopted the one-test-at-time approach are; Pairwise Independent Combinatorial Testing (PICT) [48], Automatic Efficient Test Generator (AETG) [49], Deterministic Density Algorithm (DDA) [50, 51], Test Vector Generator (TVG) [52, 53], Classification-Tree action-Tree Editor eXtended Logics (CTE-XL) [54, 55], Jenny [56], ITCH [6], GTWay [2] and Density algorithm, which is part of the DDA algorithm (extension) that authorised the DDA to support the variable strength CA. In this case, all the mentioned strategies belong to pure computational-based strategies.

The AETG was the first type of strategy using the OTAT approach to generate the test suite. AETG attempts to select a number of test case candidates in every single cycle covering most of the interactions and adds the selected test case to the final test suite in a ‘greedy’ fashion. Another greedy strategy was used for many years, where mAETG [9] and mAETG-sat [57] were extended to support the constraints. PICT [48] helps in the generation of all interactions and randomly chooses the required test cases by using the OTAT approach. Because of its random attitude, this strategy tends to offer a non-optimal test size. In addition, PICT has been able to generate test suite sizes up to $t > 6$.

As a variant of the AETG, TVG [52] produced the best results based on the three algorithms, namely; Random, Plus-one, and T-reduced sets. However, given the limited literature available, the details relating to the use of each algorithm remains indistinct. Nevertheless, based on our experience relative to the implementation of TVG, T-reduce usually generates the most optimal outcome, unlike the other related algorithms. In fact, TVG supported the interaction strength (i.e., $t < 10$) to produce the test suite.

Another strategy showing good results regarding efficiency for most configuration systems, as well as having good speed using the OTAT approach, is the Jenny strategy [56]. This strategy initially produces the test suite in order to cover one-way interaction only and was then further developed to extend the test suite to cover every 2-way interaction. This process, pending all t-way interactions, tends to be covered. The major competitor of Jenny is the Intelligent Test Case Handler, known as ITCH [6] which depends entirely on an exhaustive search algorithm for the producers of the interaction test suite. However, it is considered to be one of the slowest strategies compared to the other computational strategies regarding its efficiency to produce good results for some configuration systems. CTE_XL [54, 55] is another strategy based on the Classification-Tree Method (CTM) that utilises the OTAT approach called Classification-Tree Editor eXtended Logics (CTE_XL). The fundamental idea behind this strategy is based on several features that produce test suites by receiving the input data which is then divided into the two subsets being entirely different, and then gathering these subsets to produce the test case which is then is added to the final test suite. However, the CTE-XL strategy underpins low interaction strength for producing the test suites (i.e., $t \leq 3$). The most reliable strategy in the computational technique that uses three algorithms for producing the test suite is GTWay [2]. In this strategy, the utilised algorithms are the t-way pair generation, the parser algorithm, as well as the backtracking algorithm. In the first instance, the parser algorithm helps to ensure that the system configuration parameters are guided in order to be utilised by the algorithm of the t-way pair generation. Then, the algorithm of the t-way pair generation helps in the generation of the required t-way parameter interaction based on a symbolic representation that is defined based on the parser algorithm. Finally, the backtracking algorithm needs to iteratively go through the t-way pair sets to add up the values of the t-way parameter's interactions for the generation of a complete test t-way suite. Although, addressing the strength of the high interaction in this instance is necessary (i.e. $t \leq 12$).

Compared to the OTAT approach, the OPAT approach is where the algorithm begins initially by choosing two parameters. Then, the test suite is horizontally extended by adding the rows one-by-one until the interaction parameters have been covered, starting with a vertical extension to choose more parameters, of which the process continues until the end. One of the strategies that utilised the OPAT approach is IPO. Other strategies based on the IPO include; IPOG [58], IPOG-D [5], IPOG-F [59], IPO-s [60], and ParaOrder [61]. However, ParaOrder and IPOG strategies can support a variable strength-covering array.

C. *Heuristic Search Algorithm*

Nowadays, significant attention has been concentrated on the Heuristic search (HS) or Artificial Intelligence (AI) algorithms as part of the research interest in SBSE to solve combinatorial interaction problems. HS is a potent technique to produce the minimum array size for the test suite, although it is slow. Some of the most important heuristic search algorithms include SA [9-11, 20, 21], GA [11, 12, 22-27, 62], ACA [8, 12], PSO [7, 28-30], Cuckoo Search [3, 33, 34], Tabu Search (TS) [11, 63] and Harmony Search (HS) [4, 31, 32]. Initially, most algorithms were implemented in order to produce pairwise interaction (2-way), as by Stardom [8] such as; SA, GA and TS. The literature also shows that these strategies produce a weak result regarding efficiency given their complex structure. However, later, SA [9], GA and ACA [12] were extended to support the interaction strength (3-way). Nevertheless, the efficiency of SA is much better compared to TS and GA, and the efficiency of TS is better compared to GA.

In addition, SA, GA and TS are 'grand' strategies that support and only produce a test suite in CA. The results of the SA strategy have indicated that it is stronger compared to both, GA and ACA. Furthermore, both SA [10] and ACA [8] algorithms were extended to support the variable strength interaction up to (i.e. 3-way), where ACA is called, Ant Colony System (ACS). Searching the large space using SA to find the best test case utilised the Binary Search Algorithm (BSA). Meanwhile, ACA tried to find the best path (i.e. where every path represents a test case) until selecting the best path. In this case, the best path was selected based on the calculated weight. In GA, it begins randomly by selecting the test case after initialising the population, where each test case represented one chromosome. Then, through a fitness function, the chromosome's weight is calculated, after changing the functions (mutation and crossover) and at the end of the process, the best test case (i.e. best chromosome) is added to the final test suite. All these procedures continue until the required coverage is met. Another strategy that supports CA in by generating the test suites based on the PSO algorithm and fuzzy techniques called Fuzzy Self-Adaptive PSO (FSAPSO) [29]. FSAPSO is better compared to CS and DPSO regarding efficiency, but the performance is weak due to the Fuzzy computations. However, in this case, the strategy supported the interaction strength up to $t = 4$ to produce the final test suite.

In addition to SA and ACA [12], other strategies also are based on the GA algorithm with less modification in the structure which produces a good result regarding efficiency. These strategies are PWiesGen [25], PWiesGenPM [22], tuned Genetic (GS) [62] and Genetic Algorithm for Pairwise Test Sets (GAPTS) [26]. Notably, PWiesGenPM is better compared to GAPTS and GS is stronger than PWiesGenPM, GAPTS and PWiesGen. In order to address the variable strength problem, PWiesGen was extended [23] [to overcome this problem] with interaction strength (i.e. $t = 6$). This strategy is called PWiesGen-VSCA.

According to the literature, the first HS or AI that supports the interaction strength (i.e. $t = 6$) is the PSTG [7] strategy based on the PSO algorithm. Further, this strategy shows good performance, but weak efficiency for the strength less than $t = 3$ against the available strategies found in the literature. Calculating the weight to select the best test case to be stored needs big data by using a new approach. Accordingly, this strategy was extended to support the variable strength called VS-PSTG. Another strategy like PSTG that produces the test suite up to (i.e. $t = 6$) is the CS strategy [3], which is precisely like PSTG but is faster by using the utilised Cuckoo algorithm to reduce the search space.

One of the most efficient strategies used to produce strong results and also supports strengths (i.e. $t = 15$) is HSS [4] which is based on the Harmony Search Algorithm. Either this is where the algorithm imitates the musicians behavior, which endeavors to compose enthralling music from random sampling or from improvisations (that is, adjusting a tune from their memory). In this strategy, it adds the test case in each iteration to the test suite and then goes on until the required coverage is met. Another efficient strategy used to produce a strong result, but having less support compared to HSS regarding strengths (i.e. $t = 6$), is HHH [63]. This strategy is characterised by using High-Level Hyper-Heuristic (HHH) as the first strategy [based on literature], where it uses four algorithms (i.e. meta-heuristic algorithm) instead of one. The four algorithms are; Teaching-Learning based Optimization (TLBO), PSO, Global Neighborhood Algorithm (GNA) and the Cuckoo Search algorithm. This strategy employs Tabu Search as the high-level search, then selects one of the four algorithms in each step, and depends highly on three explicit operators, on the basis of improvement, diversification, and intensification to adaptively choose the best meta-heuristic at any point in time to produce test cases.

The test suite is also produced using the PSO family [28] according to the following strategies; DPSO, DMS-PSO, APSO, TVAC, CLPSO and CPSO. As any AI algorithm has disadvantages and advantages due to randomisation, PSO as one of the algorithms also has inherent weaknesses regarding efficiency, which is the velocity function. To address this problem, DPSO was used to produce a good result by generating the best solution. This strategy supports the interaction strength of more than 6, but the published result [28] is up to 4.

D. *Random Methods*

This method is an ad hoc approach where a random test case is selected by utilising input distribution [45]. Section *3.2* mentioned that TVG utilised three methods to test case generation. One of these methods is a random method.

IV. Overview of Artificial Bee Colony

The ABC algorithm is designed to trigger the honeybee colony foraging manner. A quintessential honeybee swarm comprised of three basic constituents e.g. source of food / employed foragers/failed recruits (onlookers and scout bees) [64]. The employed bees linked to a specific food source. They transmit important information such as (location, navigation and the profitability) of the food source and convey the data with the rest of the standby bees at the hive. The onlooker bees are responsible for food source discovery utilizing the information provided by employed bees. The scout bees assigned to random hunt the new food source. It is presumed that, the employed bees who are lacking of food source transformed into scout bees and begin a new search for the food source. It is inferred that the number of food source equates to the number of employed bees in the colony. In conclusion, the solution to the optimization problem is represented by the food source stand; meanwhile, the quality (fitness) of the mentioned solution coincides with the quantity of food source [65].

ABC initiated a random population distribution of SN solutions (food source positions) in the search space, where SN signify the size of employed or onlooker bees. Each solution X_i (i = 1, 2, …, SN) will essentially be a *D*-dimensional vector provided that the number of optimization parameters to be D. All solutions generated at this stage can be obtained from Eq. (2).

$$x_{ij} = x_{min,j} + \text{rand}(0,1)(x_{max,j} - x_{min,j}) \qquad (2)$$

Here, X_{min} and X_{max} are respectively the lower and upper boundary parameters for solution xi in dimension j (j = 1, 2 … D), and rand [0,1] is a scaling factor representing a random number between [0,1]. The D-dimensional solutions (food source positions) generated in the initialization step (C = 0) are subject to repeated cycles (C = 1, 2 …, MCN), until a termination criterion is satisfied. One ABC cycle required both implementation of global and local probabilistic search/selection. Every cycle demands various task performance by various bee types as illustrated in the flowchart (Fig. 2). The operations are initially independent and can be justified in distinct manner as below for clearer vision of the ABC methodology.

1) *Employed bee step:* After the employed bees convey the information to onlooker bees post evaluation of their sources fitness (solutions). Each employed bee agitated the old solution (X_{ij}) in its memory to create a potential solution using the Eq (3) below:

$$V_{ij} = X_{i,j} + \text{rand } [-1, 1] \, (X_{ij} - X_{kj}) \qquad (3)$$

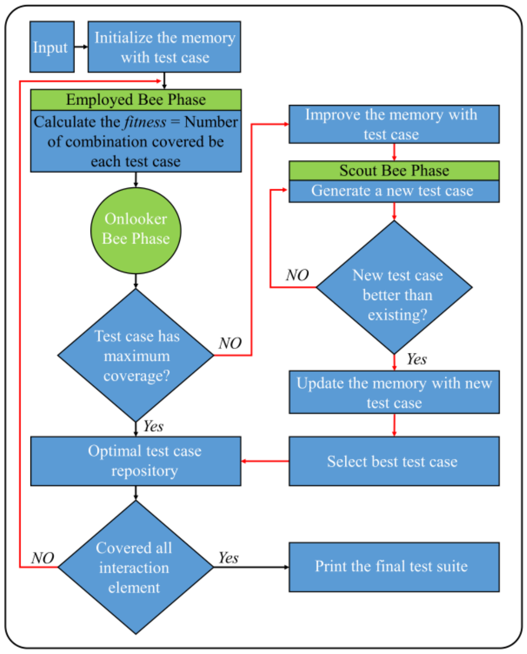
Input
Initialize the memory with test case
Employed Bee Phase
Calculate the *fitness* = Number of combination covered be each test case
Onlooker Bee Phase
Test case has maximum coverage?
NO
Yes
Improve the memory with test case
Scout Bee Phase
Generate a new test case
NO
New test case better than existing?
Yes
Update the memory with new test case
Select best test case
Optimal test case repository
Covered all interaction element
NO
Yes
Print the final test suite

Fig. 2. The ABC Algorithm Flowchart.

Here, K{1, 2, …, D} and j{1, 2, …, SN} (k s i) are randomly chosen indexes, and rand [-1,+1] is a random number between [1,1], which works as a scaling factor. Evidently, the perturbation on solution is inversely proportional to optimum solution approached in the search space. The employed bee will also evaluate the fitness of the new (perturbed) solution and according to greedy-selection scheme, supposing the fitness value is better, it will replace the old one from employed bee memory.

2) *Onlooker bee step:* The duty of onlooker bee it to search for food source (solution), in reference to the association of probability value and food source Pi. The calculation of Pi is as the expression below in Eq. 4:

$$\text{Pi} = \frac{fit_i}{\sum_{n=1}^{sn} fit_n} \qquad (4)$$

The fitness value of certain solution is signified by fit and solution number is referred to the subscript index. By comparing Pi against a randomly picked number ranging between [0, 1], the probabilistic selection is applied. The selection will be only be accepted if the random number generated is less than or equal to Pi. The approval of probabilistic selection will determine the authorization of an onlooker bee assignment to a given solution. Normally, the calculation of fitness value of solutions in problems minimization is carried as the following in Eq. 5:

$$fit_i = \begin{cases} \frac{1}{1+f_i}, & \text{if } f_{i \geq 0} \\ 1 + |f_i|, & \text{if } f_{i < 0} \end{cases} \qquad (5)$$

The objective function for solution i is signified by fi. Assuming the selected food source matches with Pi probability, with Eq. (3) the onlooker bee will select a better food source (solution) in the area of the previous one in her memory. Suppose the fitness value of the solution is better, the onlooker bee will auto update the latest solution in her mind, disregarding the old one, which is akin to employed bees.

3) *Scout bee step:* The scout bees are assigned to search random food source in order to discover better solution for the global optimization problem. The scout bees are different from employed/onlooker bees as they are not committed to old solution in order to create trial solution. They obtained their samples from a broad set of D dimensional vectors, in condition it is still within the search space zone. In ABC, the solution will be disregard if the (non-global) solution cannot be improvised post a pre-evaluated cycle's number. This will affect the assigned employed bee and transformed it to scout bee with limited scout type behavior. The limit also known as the value of this pre-evaluated cycle numbers is a crucial control factor of this algorithm. The limit is expressed as below.

$$\text{Limit} = c.n_e.D$$

Where, *Ne* signify the number of unemployed bees, *while c is* constant coefficient with a recommended value of 0.5 or 1. ABC application minimum requirement is one scout bee implementation. Scout-type operations hypothetical searches in the completely D-dimensional space provide exceptional effectiveness to the ABC method in searching the best global solution. Scout bees are independent when it comes to global optimum solution discovery in comparison to other bee types.

Both (employed/onlooker) concurrently check on their local candidate solutions for the global best. Thus, it is impossible for ABC to be trapped in local optima [64].

V. Proposed Strategy

In this study, to fulfil the optimal test suite, a new strategy is proposed. The proposed strategy uses ABC algorithm as the backbone algorithm to generate the test suites. Generally, the ABCVS strategy undergoes three phases during the construction of the uniform and variable *t*-way test suite generation using; (1) ABCVS input analysis algorithm, (2) ABCVS interaction generation algorithm and (3) generating the test suite using ABC algorithm as shown in Fig. 3. In Section 5.1, it discusses how to set up the input for the next phase and describes the generating of the interaction tuples by using the interaction generation algorithm. The ABCVS test suite construction is described in Section 5.2.

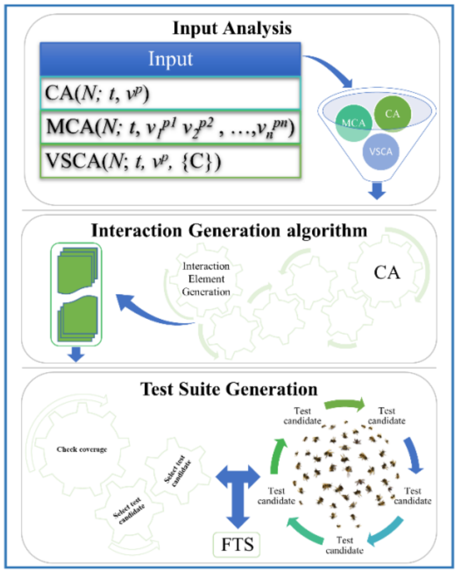

Fig. 3. ABCVS Strategy Design.

A. Input Analysis & Interaction Generation Element Algorithm

This section illustrates how the strategy receiving the CA, MCA and VSCA (inputs) and generation of the interaction parameter (*P*) combinations, compute and store the interaction element for each of the parameters to be used later for the test case coverage evaluation; based on the values (*v*) and the specified interaction strengths (*t*). To clarify the input analysis and the interaction generation elements, Fig. 4 shows a small VSCA configuration as an example illustrated by using a flowchart.

The value interaction elements of each parameter are then constructed based on the parameter interactions. As shown in the previous example (N; 2, 2^3 3^1, {CA (3, 2^3)}), the main configuration has four parameters, where the first three parameters have two values (0 and 1), and the fourth parameter has three values (0, 1, and 2) with interaction strength t_m = 2. The sub-configuration has three parameters, each having two values (0 and 1) with interaction strength t_s = 3. As illustrated in Fig. 4, the generation approach for the main configuration (t_m = 2) has six possible combinations of the interaction's parameters. For example, the possibility of the interaction elements for combinations (0 B2 0 B4) is 2 ×3 for both the second and fourth parameters, and the possibility of the interaction elements for combinations (B1 0 B3 0) is 2 ×2 between the first and third parameters. In order to check the availability of the parameters, the algorithm scans the binary digit of the combination, where each value of the other parameters is included in the interaction element. Regards to the rest of the values (i.e., replaced with "X") indicates "don't care" is the excluded value in the interaction element. The interaction elements of the sub-configuration were produced similarly to that as depicted in Fig. 4. Generating the interaction elements for each test case continues until creating the final test suite. This method is illustrated in the next section.

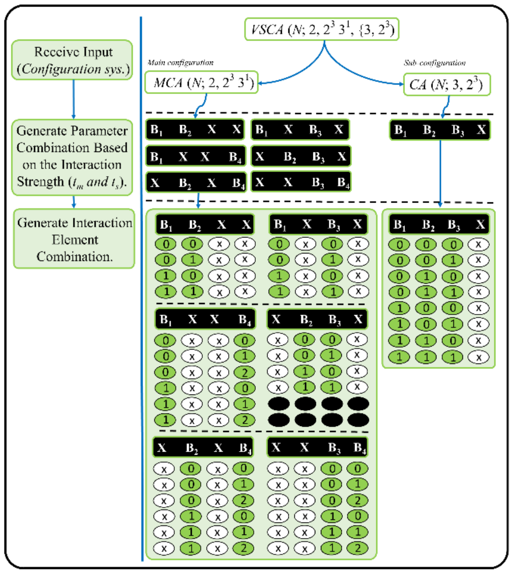
VSCA (N; 2, 2^3 3^1, {3, 2^3})
Receive Input (Configuration sys.)
Main configuration
MCA (N; 2, 2^3 3^1)
Sub configuration
CA (N; 3, 2^3)
Generate Parameter Combination Based on the Interaction Strength (t_m and t_s).
Generate Interaction Element Combination.
B1 B2 X X
B1 X B3 X
B1 X X B4
X B2 B3 X
X B2 X B4
X X B3 B4
B1 B2 B3 X

Fig. 4. An Illustration Example for the ABCVS Strategy.

B. The Test Suite Generation Algorithm

According to Fig. 3 that showed the responsible algorithm for the parameter combination generator, and the interaction elements combination generator based on the interaction strength, this will generate the possible interaction elements. In this strategy, a unique answer is produced in each step of the algorithm, which has no effect on the next test case selection. Selecting a test case is the most challenging problem in test suite generation, to achieve the test objective with a minimum number of test cases. In ABCVS, each procedure in this algorithm generates the best solution, which does not affect the next test case selection. For example, at every stage, the first of each bee (employed and onlooker) is randomly valued by the scout bee to the number of food source (which represents a test case) within a particular area. Following the valuation, the quality of food source is called the fitness value (coverage or weight) which is calculated based on the fitness function, and the results are presented to the employed bees. The employed bees optimise the solutions (food source position), calculate their qualities again, and present the results to the onlooker bees. The onlooker bees calculate the probability of selecting the solutions. Then, these values are normalised, and the food source with higher quality (fitness) are greedily selected, and the proposed solutions are optimised again by the random values and the values of neighbors. The quality of the proposed solutions based on the fitness function is again specified. The onlooker bees investigate the solutions and select the best solution for storing. If it is an obsolete solution, the scout bee generates a new one. Otherwise, if the algorithm is finished, the final solution is registered as the output of the first stage, if not the algorithm should repeat. For instance, in Fig. 5, the test case is selected based on the maximum fitness (coverage or weight). The test case (1000) has the maximum weight of coverage, which is seven. Therefore, this test case (1000) adds to the final test suite by the test suite generation algorithm. Regarding the test case (1001) that does not reach the maximum coverage of fitness, the algorithm updates the population (food source) search space randomly to obtain the best fitness.

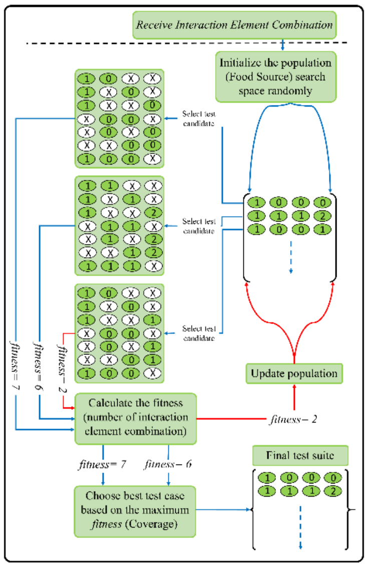
Receive Interaction Element Combination
Initialize the population (Food Source) search space randomly
Select test candidate
Select test candidate
Select test candidate
Update population
Calculate the fitness (number of interaction element combination)
fitness = 7
fitness = 6
fitness = 2
Final test suite
Choose best test case based on the maximum fitness (Coverage)

Fig. 5. An Illustration Example for Selection Test Case based on the Maximum Coverage.

VI. Tuning Parameter Setting of ABCVS

There are three important control parameters in the proposed ABCVS strategy. These control parameters are the colony size number (employed bee and onlooker bee (population size)) equal to the number of food source, the value of *Limit* and the maximum cycle number (MCN). Growing the population size improves the outcome regarding both the generation time and the array size. In ABCVS, generating the final test suite plays a critical role based on the employed and onlooker phase. It is worth mentioning that these two phases are different in determining the value of configurations. In other words, both of *t*, *p* and *v* affect the number of iterations for the employed and onlooker phase.

An experiment were conducted on CA (N; 2, 5^7) with a variable number of the bee to determine this value. As shown in Fig. 6 conducted, the best value of this configuration for the number of bees is 1 (i.e. food source = population /2) when the number of cycles (70) is a test suite of size 38 test cases, and when the number of bees is 2, the number of cycles (100) is 37. Increasing the value of bees up to 3 and 4, when the number of cycles is (100) and (90), respectively, the algorithm achieves a test case of size 36 test cases. In these experiments, the *Limit* is kept constant = 100. Therefore, the value increasing does not affect the size, only increasing in the generation time. In Fig. 6, different colours that depend on the number of cycles show the size of the test cases.

Fig. 7 illustrates the effects of the colony size (population size) on the array size in the same configuration CA (N; 2, 5^7), and shows the best average test suite size for CA (N; 2, 5^7). The population of size is variable from 1 to 10, and 20 to 100, when the number of cycles also is variable from 10 to 100 and *Limit* = 100 is considered. Regarding the test suite array size analysis, Meta-heuristic is a non-deterministic algorithm based on randomisation. Therefore, this analysis has two values. First, the best value as illustrated in Fig. 6, and the second value is the average of five independent runs. The best average is shown in Fig. 7 for the configuration CA (N; 2, 5^7), where the best value for this configuration is 36 test cases.

The analysis, in this case, is aimed at determining the suitable value for the colony size (population size) that depends on changing the values of *p*, *t* and *v* of the configuration. Each configuration then determines the suitable experiments. Therefore, the best average value is selected when the population size is 50, and the maximum number of cycles is 80. Note that this value is related to the configurations in the tables of this paper.

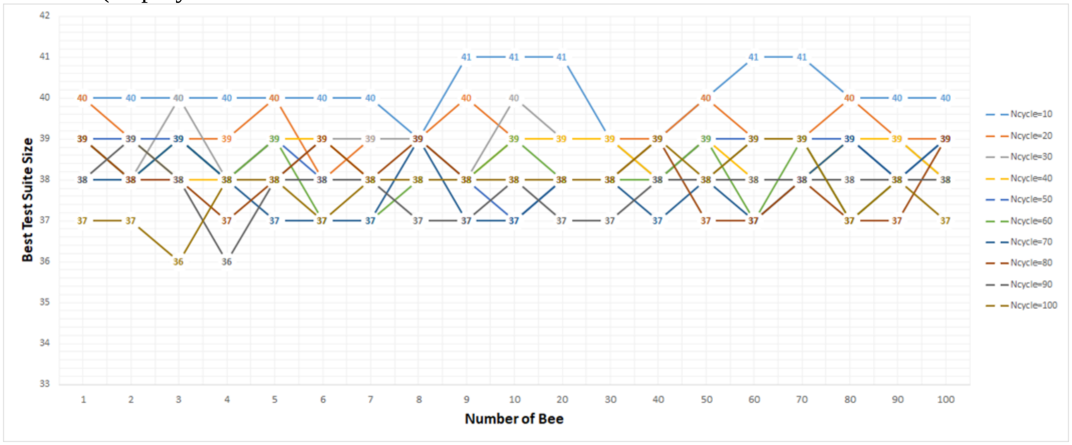

Fig. 6. The Best Test Suite Size with Variant Number of Bee for CA (N; 2, 5^7).

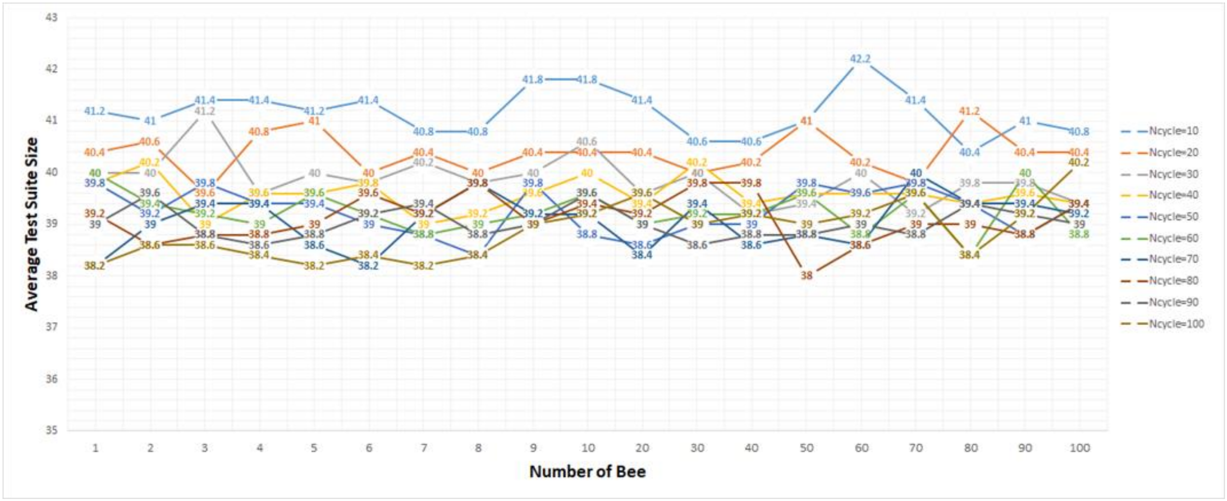

Fig. 7. The Best Average Test Suite Size with Variant Number of Bee for CA (N; 2, 5^7).

VII. Evaluation

In this section, there are two parts in the evaluation of the test suite generation strategies; (1) the efficiency evaluation (i.e. test suite size) and (2) the performance evaluation (i.e. the time of test suite generation) [62]. The test suite size and the interaction strength are not affected by the operating system and hardware but depend on the steps of the algorithm's running. Different strategies are selected in different tables based on the criterion; Tables IV-VIII are divided into two categories. In the first category, the available strategies that can be implemented in our system are included regarding efficiency; ABCVS is compared with these strategies. The second category contains unavailable strategies as discussed regarding efficiency and performance by other researchers. The comparison of these strategies has been conducted based on efficiency and performance. Therefore, it implements some of the AI strategies since many are unavailable in the same conditions of the proposed strategy (i.e. hardware and software). Regarding the Tables IV-VIII of configurations, the mentioned strategy's values are available, and used the same value as that published in the papers; otherwise, implementing the strategies on the system is used to obtain the result.

The characteristics of the hardware system for conducting the experiments of the proposed strategy consisted of a Windows 7 (OS) desktop computer with 3.40 GHz Xeon (R) CPU E3 and 8GB RAM. The Java language JDK 1.8. was used to code and implement the ABCVS. In this paper, the best outcome in the tables is shown in bold and in dark cells. Also, “NA” (Not Available) means not publishing, and “NS” (Not Supported) means does not support the corresponding configuration.

A. *Evaluation of the Efficiency*

In Table IV, the array size evaluation is divided into five parts; the interaction strength $2 \leq t \leq 6$ and parameter $3 \leq p \leq 12$ and value and $V = 3$ for configuration CA (N; *t*, *p*, *v*). The outcome of IPOD-D and IPOG are adopted from the Advanced Combinatorial Testing System (ACTS) [66]. These strategies do not normally generate good results as well as for PICT, TConfig and Jenny. Whereas, IPOG-D, IPOG and TConfig support generation of the interaction strength up to $t = 6$. As shown in Table VII, other AI-based strategies are studied. The results of CS and PSTG are unavailable and are obtained from the papers. CS and PSTG generate similar results. The strongest strategy in terms of producing the best result is DPSO, where it showed results for $t \leq 4$ obtained from [24], for each configuration of 30 separate runs. Regarding the weak performance of DPSO (i.e. speed), it is impossible to implement each configuration of 30 runs because in CA (2512; 6, 3^{12}) 23,620 s are required, thus will take more than 7 days. Other results are available from [67]. The result for $t > 4$ is obtained with 3 runs only. CS another AI-based strategy has generated a good result. In this case, the obtained result for $t > 4$ is adopted in 3 runs only because of the low performance of this strategy. To be fair, the ABCVS strategy is adopted in 5 runs for $t > 4$, and 20 runs for the value $t \leq 4$.

The values in Tables V, VI and VII, are adopted from the corresponding paper [7]. Regarding the configuration system VSCA (N; 2, 3^{15}, {C}) in Table V, where the PICT and WHITCH strategy produces undesirable results, these strategies have the capability to generate a test suite to $t = 6$. However, WHITCH produced better results. Another computational-based strategy is ParaOrder that had a better result compared to PICT and WHITCH up to $t = 3$ but does not support in generating a test suite for $t \geq 4$. As shown in this table, AI-based strategies are considered the strongest strategies. The strongest strategies that generate the best results are SA and ACS when the strength is 3. However, these strategies are not able to produce a test suite of more than 3. Regarding the strategies such as PwiseGen-VSCA, ABCVS, GS and PSTG, these are better than the other strategies in terms of support strength of more than 3 and less than 6. IPOG and GS have the support strength more than 6.

Table VI shows the configuration VSCA (N; 3, 3^{15}, {C}), where PSTG, GS and ABCVS strategies generate good results for strength up to $t = 6$, but IPOG and GS produce a better test suite for strength more than 6. However, ACS and SA are not able to generate a result due to not having support strength of more than 3. Also, the results for Density and ParaOrder are not available.

Regarding the configuration system VSCA (N, 2, 4^3 5^3 6^2, {C}) in Table VII, the results of PwiseGen-VSCA and GS strategies are not available as well as for the ParaOrder and Density strategies for most subsets ({C}), but not all. Furthermore, WHITCH and PICT have support strength up to 6, but these strategies do not generate a good result. ABCVS, TVG and PSTG strategies have the capability to generate better results for strength *t* = 6, and ACS and SA produced a better result for strength *t* = *3*.

TABLE IV. A TEST SUITE SIZE FOR CA (N; T, 3^P) WITH 2 ⩽ T ⩽ 11 AND 3 ⩽ P ⩽ 12

t	P	Pure computation strategies					AI-based strategies					
		Jenny Best	TConfig Best	PICT Best	IPOG-D Best	IPOG Best	DPSO Best	PSTG Best	CS Best	GS Best	ABCVS Best	ABCVS Avg.
2	3	**9**	10	10	15	**9**	**9**	**9**	**9**	**9**	**9**	9.95
	4	13	10	13	15	**9**	**9**	**9**	**9**	**9**	**9**	10.70
	5	14	14	13	15	15	**11**	12	**11**	**11**	**11**	12.75
	6	15	15	14	15	15	14	**13**	**13**	**13**	**13**	14.75
	7	16	15	16	15	15	**14**	15	**14**	**14**	15	15.50
	8	17	17	16	**15**	**15**	**15**	**15**	**15**	**15**	**15**	15.90
	9	18	17	17	**15**	**15**	**15**	17	16	**15**	16	17.25
	10	19	17	18	21	**15**	16	17	17	16	17	17.70
	11	17	20	18	21	17	17	17	18	**16**	17	18.45
	12	19	20	19	21	21	**16**	18	18	**16**	18	19.25
3	4	34	32	34	**27**	32	**27**	**27**	28	**27**	**27**	33.50
	5	40	40	43	45	41	41	39	**38**	**38**	**38**	41.45
	6	51	48	48	45	46	**33**	45	43	43	44	46.85
	7	51	55	51	50	55	**48**	50	**48**	49	49	51.90
	8	58	58	59	**50**	56	52	54	53	54	54	55.85
	9	62	64	63	71	63	**56**	58	58	58	58	59.80
	10	65	68	65	71	66	**59**	62	62	61	62	64.25
	11	65	72	70	76	70	**63**	64	66	**63**	66	68.15
	12	68	77	72	76	73	**65**	67	70	67	70	72.10
4	5	109	97	100	162	97	**81**	96	94	90	98	103.65
	6	140	141	142	162	141	131	133	132	**129**	135	138.75
	7	169	166	168	226	167	**150**	155	154	153	157	161.45
	8	187	190	189	226	192	**171**	175	173	173	179	182.05
	9	206	213	211	260	210	**187**	195	195	194	197	200.95
	10	221	235	231	278	233	**206**	210	211	209	215	217.90
	11	236	258	249	332	251	**221**	222	229	223	234	236.50
	12	252	272	269	332	272	237	244	253	**236**	251	254.20
5	6	348	305	310	386	305	**244**	312	304	301	274	317.70
	7	458	477	452	678	466	**438**	441	434	432	442	449.95
	8	548	583	555	756	575	517	**515**	**515**	**515**	530	534.20
	9	633	684	637	1043	667	591	598	**590**	594	609	613.50
	10	714	773	735	1118	761	**667**	**667**	682	672	688	690.60
	11	791	858	822	1372	851	**735**	747	778	741	762	765.50
	12	850	938	900	1449	929	**802**	809	880	806	814	817.71
6	7	1089	921	1015	1201	921	**729**	977	973	963	944	984.35
	8	1466	1515	1455	1763	1493	1409	1402	1401	**1399**	1424	1438.6
	9	1840	1931	1818	2526	1889	1682	1684	1689	**1681**	1756	1767.0
	10	2160	>day	2165	2834	2262	**1972**	1980	2027	1980	2055	2060.1
	11	2459	>day	2496	3886	2607	**2250**	2255	2298	2258	2261	2269.2
	12	2757	>day	2815	4087	3649	**2512**	2528	2638	2558	2571	2576.5

TABLE V. TEST SUITE SIZE FOR VSCA (N, 2, 3^{15}, {C})

{C}	Pure computation strategies						AI-based strategies						
	WHITCH Best	IPOG Best	ParaOrder Best	Density Best	TVG Best	PICT Best	SA Best	ACS Best	PSTG Best	GS Best	PwiseGen Best	ABCVS Best	ABCVSAvg.
Ø	31	21	33	21	22	35	16	19	19	19	16	20	21.3
CA (3, 3^3)	48	27	27	28	27	81	27	27	27	28	27	27	27.85
CA (3, 3^4)	59	39	27	32	35	105	27	27	30	29	27	32	35.1
CA (3, 3^5)	62	39	45	40	41	131	33	38	38	38	33	41	42.9
CA (4, 3^4)	103	81	NA	NA	81	245	NA	NA	81	81	81	81	81.2
CA (4, 3^5)	118	122	NA	NA	103	301	NA	NA	97	92	91	**90***	100.35
CA (4, 3^7)	189	181	NA	NA	168	505	NA	NA	158	155	158	**154***	160.2
CA (5, 3^5)	261	243	NA	NA	243	730	NA	NA	243	243	243	243	243.1
CA (5, 3^7)	481	581	NA	NA	462	1356	NA	NA	441	441	441	446	449.2
CA (6, 3^6)	745	729	NA	NA	729	2187	NA	NA	729	729	729	729	729
CA (6, 3^7)	1050	967	NA	NA	1028	3045	NA	NA	966	960	NA	956	961.1
CA (3, 3^4) CA (3, 3^5) CA (3, 3^6)	114	51	44	46	53	1376	34	40	45	NA	NA	82	85.1
CA (3, 3^6)	61	53	49	46	48	146	34	45	45	46	40	45	46.7
CA (3, 3^7)	68	58	54	53	54	154	41	48	49	50	47	50	51.85
CA (3, 3^9)	94	65	62	60	62	177	50	57	57	57	57	58	60.1
CA (3, ^{15})	132	NS	82	70	81	83	67	76	74	75	74	81	83.2

TABLE VI. TEST SUITE SIZE FOR VSCA (N, 3, 3^{15}, {C})

{C}	Pure computation strategies						AI-based strategies						
	WHITCH Best	IPOG Best	ParaOrder Best	Density Best	TVG Best	PICT Best	SA Best	ACS Best	PSTG Best	GS Best	PwiseGen Best	ABCVS Best	ABVCS Avg.
Ø	75	82	NA	NA	84	83	NS	NS	75	74	NA	81	83.25
CA(4, 3^4)	129	87	NA	NA	93	1507	NS	NS	91	88	NA	93	96.1
CA(5, 3^5)	273	243	NA	NA	244	5366	NS	NS	243	243	NA	243	246.2
CA(6, 3^6)	759	729	NA	NA	729	12,609	NS	NS	729	729	NA	729	729
CA(4, 3^5)	151	119	NA	NA	118	1793	NS	NS	114	111	NA	115	118.6
CA(5, 3^6)	387	337	NA	NA	323	5387	NS	NS	314	308	NA	316	329.15
CA(6, 3^7)	1441	1215	NA	NA	1018	16,792	NS	NS	1002	959	NA	**949***	956.6
CA(4, 3^7)	219	183	NA	NA	168	2781	NS	NS	159	158	NA	**157***	161.9
CA(4, 3^9)	289	227	NA	NA	214	3095	NS	NS	195	194	NA	196	199.9
CA(4, ^{11})	354	259	NA	NA	256	2824	NS	NS	226	226	NA	333	237.0
CA(4, ^{15})	498	498	NA	NA	327	NA	NS	NS	284	282	NA	308	441
CA(5, 3^7)	481	713	NA	NA	471	7475	NS	NS	437	437	NA	439	448.7
CA(5, 3^8)	620	714	NA	NA	556	8690	NS	NS	516	516	NA	527	535.65
CA(6, 3^8)	1513	2108	NA	NA	1479	22,833	NS	NS	1396	1397	NA	1424	1436
CA(6, 3^9)	1964	2124	NA	NA	1840	26,729	NS	NS	1690	1687	NA	1752	1763

TABLE VII. TEST SUITE SIZE FOR VSCA (N, 2, $4^3\ 5^3\ 6^2$, {C})

{C}	Pure computation strategies						AI-based strategies						
	WHITCH Best	IPOG Best	ParaOrder Best	Density Best	TVG Best	PICT Best	SA Best	ACS Best	PSTG Best	GS Best	PwiseGen Best	ABCVS Best	ABCVS Avg.
Ø	48	43	49	41	44	43	36	41	42	NA	NA	44	44
CA (3, 4^3)	97	83	64	64	67	384	64	64	64	NA	NA	64	64
CA (3, $4^3\ 5^2$)	164	147	141	131	132	781	100	104	124	NA	NA	128	128
CA (3, 5^3)	145	136	126	125	125	750	125	125	125	NA	NA	125	125
CA (4, $4^3\ 5^1$)	354	329	NA	NA	320	1920	NS	NS	320	NA	NA	320	320
CA (5, $4^3\ 5^2$)	1639	1602	NA	NA	1600	9600	NS	NS	1600	NA	NA	1600	1600
CA (3, 4^3) CA (3, 5^3)	194	136	129	125	125	8000	125	125	125	NA	NA	125	125
CA (4, $4^3\ 5^1$) CA (4, $5^2\ 6^2$)	1220	900	NA	NA	900	288,000	NS	NS	900	NA	NA	900	900
CA (3, 4^3) CA (4, $5^3\ 6^1$)	819	750	NA	NA	750	48,000	NS	NS	750	NA	NA	750	750
CA (3, 4^3) CA (5, $5^3\ 6^2$)	4569	4500	NA	NA	4500	288,000	NS	NS	4500	NA	NA	4500	4500
CA (4, $4^3\ 5^2$)	510	512	NA	NA	496	2874	NS	NS	472	NA	NA	**463***	463
CA (5, $4^3\ 5^3$)	2520	2763	NA	NA	2592	15,048	NS	NS	2430	NA	NA	**2403***	2403
CA (3, $4^3\ 5^3\ 6^1$)	254	215	247	207	237	1266	171	201	206	NA	NA	213	213
CA (3, $5^1\ 6^2$)	188	180	180	180	180	900	180	180	180	NA	NA	180	180
CA (3, $4^3\ 5^3\ 6^2$)	312	NS	307	256	302	261	214	255	260	NA	NA	266	266

TABLE VIII. TEST SUITE SIZE AND TIME FOR CA (N; T, 7, 3^7) WITH $2 \leqslant T \leqslant 7$

t	Pure computation strategies					AI-based strategies					
	Jenny N/Time	TConfig N/Time	PICT N/Time	IPOG-D N/Time	IPOG N/Time	DPSO N/Time	HSS N/Time	PSO N/Time	CS N/Time	GS N/Time	ABCVS N/Time
2	16/0.04	15/0.08	16/0.01	18/**0.001**	15/**0.001**	**14**-Mar	**14**/0.92	15/2.2	15/0.28	**14**/0.22	**14**/2.1
3	51/0.09	55/0.43	51/0.04	50/**0.001**	55/**0.001**	**48**/23	50/4.02	50/8.2	50/3.1	49/0.78	49/23.07
4	169/0.3	166/8.24	168/0.09	226/**0.001**	167/**0.001**	**150**/156	154/23.1	157/33.2	156/6.2	153/3.01	157/148.7
5	458/0.72	477/72.96	452/0.7	678/0.062	466/**0.001**	438/191	438/70.8	439/113	436/19.1	**432**/8.85	439/457.9
6	1087/1.10	921/425.52	1015/1.20	1201/0.062	921/**0.001**	**729**/147	926/107.8	981/382	973/31	963/16.98	862/343.5

TABLE IX. TEST SUITE SIZE AND TIME FOR CA (N; 3, 3^P) WITH $4 \leqslant P \leqslant 20$

P	Pure computation strategies					AI-based strategies					
	Jenny N/Time	TConfig N/Time	PICT N/Time	IPOG-D N/Time	IPOG N/Time	DPSO N/Time	HSS N/Time	PSO N/Time	CS N/Time	GS N/Time	ABCVS N/Time
4	34/0.01	32/0.07	34/0.04	**27**/**0.001**	32/**0.001**	**27**-Feb	30/1.9	28/4.2	**27**/1.30	**27**/0.40	31/1.1
5	40/0.03	40/0.10	43/0.09	45/**0.001**	41/**0.001**	41/7	39/2.7	39/6.1	**38**/2.09	**38**/0.63	40/3.2
6	51/0.09	48/0.31	48/0.13	45/**0.001**	46/**0.001**	**33**/10	44/3.2	45/7.5	45./2.63	43/0.73	45/9.6
7	51/0.07	55/0.43	51/0.23	50/**0.001**	55/**0.001**	**48**/23	50/4.02	50/8.2	50/3.12	49/0.78	49/24.7
8	58/0.07	58/1.23	59/0.36	**50**/**0.001**	56/**0.001**	52/36	54/4.8	54/9.3	55/4.04	54/0.86	55/55.3
9	62/0.08	64/1.72	63/0.57	71/**0.001**	63/**0.001**	**56**/55	59/6.01	58/10.5	60/4.69	58/1.11	59/117.8
10	65/0.10	68/2.84	65/0.64	71/**0.001**	66/**0.001**	**59**/81	62/7.3	62/11.3	64/5.60	61/1.24	63/248.2
11	65/0.12	72/3.93	70/0.70	76/**0.001**	70/**0.001**	**63**/115	66/9.6	64/12.8	66/7.12	**63**/1.28	67/534.4
12	68/0.18	77/5.24	72/0.79	76/**0.001**	73/**0.001**	**65**/157	67/11.5	67/13.6	70/8.41	67/1.53	71/702.7

TABLE X. TEST SUITE SIZE AND TIME FOR CA (N; 3, v^7) WITH $2 \leqslant v \leqslant 6$

v	Pure computation strategies					AI-based strategies					
	Jenny N/Time	TConfig N/Time	PICT N/Time	IPOG-D N/Time	IPOG N/Time	DPSO N/Time	HSS N/Time	PSO N/Time	CS N/Time	GS N/Time	ABCVS N/Time
2	14/0.04	16/0.03	15/0.09	14/**0.001**	16/**0.001**	15-Jun	**12**/0.9	**12**/1.7	13/0.82	**12**/0.19	**12**/3.9
3	51/0.09	55/0.43	51/0.19	50/**0.001**	55/**0.001**	**48**/24	50/4.02	50/8.2	50/2.1	50/0.78	49/23.3
4	124/0.12	**112**/2.57	124/0.53	**112**/**0.001**	124/**0.001**	**112**/54	121/5.9	118/21.7	119/6.6	117/1.83	119/97.1
5	236/0.61	239/03.18	241/0.78	252/**0.001**	237/**0.001**	**216**/172	233/18.5	226/43.2	233/14.4	231/3.63	230/368.6
6	400/1.00	423/13.15	413/1.00	470/**0.001**	420/**0.001**	**365**/188	411/31.3	420/88.6	403/18.2	397/6.48	394/1076.2

B. *Evaluation of the Performance*

In Table VIII, the configuration CA (N; *t*,3^7) for 2 ≤t ≤6 is used in terms of array size and time. TConfig strategy speed relies on *t*, where by increasing the *t* time will increase exponentially. IPOG-D and IPOG are fastest computational-based strategies, where the test suite generation time almost near to zero that indicates the high performance. ABCVS and DPSO performance are very slowly. PICT, Jenny and GS are faster than ABCVS, but these strategies are not able to generate the final test suite in a less than day. Therefore, the proposed strategy it's better in terms of performance than PICT, Jenny and TConfig. The result of the proposed strategy shows the generation time of a test suite does not really rely on increasing *t*.

Table IX displays the results in terms of array size and time of the strategies. The configuration CA (N; 3, 3^P) is used to evaluate the variable of p on time generation. The growth of p has shown an impact on the proposed strategy in terms of time generation, but has less impact on TConfig performance.

The last evaluation in terms of time generation, the configuration CA (N; 3, 7, v) for 2 ≤ v ≤ 10 is used in Table X. In this evaluation will test the impact of increasing the values on time generation. IPOG and IPOG-D are the fastest strategies in terms of the performance and DPSO is strongest on terms of array size. ABCVS generates close results to DPSO and GS.

VIII. STATISTICALLY EVALUATION

In order to evaluate the strategy regarding array size (i.e. effectiveness), a statistical method is another way to assess the significance of strategy. In this case, the Wilcoxon signed-rank test is used between ABCVS and each strategy in the experimental tables 95% confidence level (i.e. α =0.05). The reason for adopting this method is that the Wilcoxon signed-rank test takes into account the difference between two sets. To study the difference of the two sets, this test is ideal. In other words, this test is measured from a subject group and can be rated.

Due to the multiple comparisons, we need to control the error rate. Bonferroni-Holm correction was adopted to adjusting α value (i.e. in other words, based on Holm's sequentially rejective step down procedure [26]) depending on the first stored p-value (Asymp. Sig. (2-tailed)) in scaling in ascending order. Therefore, α Holm is adjusted based on:

$$\alpha \text{ Holm} = \frac{\alpha}{M-i+1}$$

Note: Where M indicates the overall number of paired comparison and i indicates the test number.

The test is executed using a software tool called SPSS, where if the value is less than α Holm of the Asymp. Sig. (2-tailed), it indicates a significant difference between the two sets. There are four values to evaluate ABCVS; Ranks ABCVS>, ABCVS<, and ABCVS= are used. In other words, the results of the proposed strategy are greater, smaller or equal to the other existing strategies. Two values have a Statistical Test part; Asymp. Sig. (2-tailed) and Z. The value of Asymp. Sig. (2-tailed) indicates the significant difference between the two sets and that the value does not exceed α Holm. Regarding the value Z, it is out of the scope of this paper (i.e. not important). The strategies with N/A and N/S results are considered incomplete and ignored samples, as there is no available result for the specified test configuration.

Tables XI–XIV present a statistical test for the Tables IV–VII and Table XI presents the result of Table IV for the Wilcoxon test. ABCVS shows there is a significant difference with other strategies in column Asymp. Sig. (2-tailed), except for CS, which has a significant difference from ABCVS.

Table XII presents the test results of Table V. IPOG, ParaOrder, Density, TVG, GS, SA, ACS and PwiseGenVSCA produce a test suite for strength less than 3 for *t* > 3 and the results are considered "missing". For this reason, the IPOG, ParaOrder, Density, TVG, PSTG and GS results are better than ABCVS. Whereas, ABCVS performs better than WHITCH, PICT, SA, ACS and PwiseGen-VSCA in this table.

Table XIII presents the test results of Table VI, where ABCVS excelled compared to WHITCH and GS strategies. Table XIV presents the test results of Table VII. TVG and PSTG are shown to have a significant difference compared to ABCVS. However, ABCVS excelled with the WHITCH, and PICT strategies.

IX. DISCUSSION

AI-based strategies are characterised by producing a strong result, although, not without having a complex structure and repetition of which the complex structure of the strategy is inversely correlated with the strategy's strength. For instance, ACO, GA and SA can produce a test suite with *t* <= 3. While CS and PSTG reduce complexity by changing the structure and raising the performance speed to support strengths up to *t* = 6. Notably, GS supports strengths up to *t* = 15.

IPOG, PICT and Jenny are computational based strategies having the capability to produce a test suite for $t > 6$ like AI-based strategies. However, computational strategies do not produce good results but instead, do have good performance. Therefore, it is impossible to obtain a strategy that can support high interaction strength with perfect performance and efficiency. The ABCVS indicates that its efficiency regarding generating a test suite with $t = 6$ competes well compared to the other existing strategies. Further, ABCVS is better as compared to computational ones and can compete with AI-based strategies regarding performance and efficiency.

Although, ABCVS like any other strategy have limitations; for example, its contribution towards supporting variable strengths interaction. The ABCVS efficiency for strengths (i.e. $t \leq 3$) is slightly lower compared to others. Given the limited literature in this field, many of these strategies are not available publicly and therefore, in this study, the configurations reported and available in publications have been used. Therefore, further experiments need to be conducted in order to obtain a precise evaluation of the proposed strategy.

TABLE XI. WILCOXON SIGNED RANK SUM TEST FOR TABLE IV.

Pairs	**Ranks**			**Test statistics**			**Conclusion**
	ABCVS >	**ABCVS <**	**ABCVS =**	**Z**	**Asymp. Sig. (2-tailed)**	**α holm**	
ABCVS- Jenny	2	36	2	-5.199269	0.00000020	0.00625000	Reject
ABCVS- PICT	0	40	0	-5.522230	0.00000003	0.00714286	Reject
ABCVS- IPOG-D	3	34	3	-5.107580	0.00000032	0.00833333	Reject
ABCVS- IPOG	4	31	5	-4.662525	0.00000300	0.01000000	Reject
ABCVS- DPSO	34	2	4	-4.999589	0.00000057	0.01250000	Reject
ABCVS- PSTG	22	5	13	-3.236855	0.00120900	0.01666667	Reject
ABCVS- CS	20	8	12	-1.071752	0.28383200	0.02500000	Retain
ABCVS- GS	31	2	7	-3.966042	0.00007300	0.05000000	Reject

TABLE XII. TWILCOXON SIGNED RANK SUM TEST FOR TABLE V.

Pairs	**Ranks**			**Test statistics**			**Conclusion**
	ABCVS >	**ABCVS <**	**ABCVS =**	**Z**	**Asymp. Sig. (2-tailed)**	**α holm**	
ABCVS- WHITCH	0	16	0	-3.518549	0.000434	0.01250000	Reject
ABCVS- TVG	1	9	6	-1.888148	0.059006	0.01666667	Retain
ABCVS- PICT	0	16	0	-3.516196	0.000438	0.02500000	Reject
ABCVS- PSTG	8	3	5	-0.757616	0.448681	0.05000000	Retain

TABLE XIII. WILCOXON SIGNED RANK SUM TEST FOR TABLE VI.

Pairs	**Ranks**			**Test statistics**			**Conclusion**
	ABCVS >	**ABCVS <**	**ABCVS =**	**Z**	**Asymp. Sig. (2-tailed)**	**α holm**	
ABCVS- WHITCH	1	14	0	-3.353003	0.000799	0.01000000	Reject
ABCVS- IPOG	2	11	2	-2.480941	0.013104	0.01250000	Retain
ABCVS- TVG	1	12	2	-2.341884	0.019187	0.01666667	Retain
ABCVS- PSTG	11	2	2	-2.103645	0.035409	0.02500000	Retain
ABCVS- GS	11	2	2	2.551605	0.010723	0.05000000	Reject

TABLE XIV. WILCOXON SIGNED RANK SUM TEST FOR TABLE VII.

Pairs	**Ranks**			**Test statistics**			**Conclusion**
	ABCVS >	**ABCVS <**	**ABCVS =**	**Z**	**Asymp. Sig. (2-tailed)**	**α holm**	
ABCVS- WHITCH	0	15	0	-3.410523	0.000648	0.01250000	Reject
ABCVS- TVG	0	6	9	-2.201398	0.027708	0.01666667	Retain
ABCVS- PICT	2	13	0	-3.237382	0.001206	0.02500000	Reject
ABCVS- PSTG	4	2	9	-0.104828	0.916512	0.05000000	Retain

X. Conclusion

This paper proposes an efficient strategy called ABCVS, based on the ABC algorithm for both uniform and variable CAs. Supporting variable strength is the main contribution of ABCVS. In addition to the supporting variable strength, ABCVS can generate a test suite up to $t = 6$ and can produce a good result with suitable performance. To study the impact of parameters like population size or a number of cycles, different experiments have been conducted. The suitable tuning parameters of the proposed ABCVS strategy results in better ABCVS, regarding higher interaction, performance and efficiency. Furthermore, different experiments have been conducted on different configurations to compare ABCVS with existing strategies, where ABCVS shows it can compete against the other strategies regarding both efficiency and performance. As part of our future work, we want to study other metaheuristic approaches to hybrid these with ABC to increase efficiency. This hybridisation should be performed in a way that does not decrease performance and can increase the support for test suite generation for $t > 6$.

References

[1] Kalaee, A. and V. Rafe, An optimal solution for test case generation using ROBDD graph and PSO algorithm. Quality and Reliability Engineering International, 2016. 32(7): p. 2263-2279.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] Hartman, A., T. Klinger, and L. Raskin, IBM intelligent test case handler. http://www.alphaworks.ibm.com/tech/whitch, 2016. 284(1): p. 149-156.

[7] Ahmed, B.S., K.Z. Zamli, and C.P. Lim, Application of particle swarm optimization to uniform and variable strength covering array construction. Applied Soft Computing, 2012. 12(4): p. 1330-1347.

[8] Chen, X., et al., Variable strength interaction testing with an ant colony system approach, in Software Engineering Conference, 2009. APSEC'09. Asia-Pacific. 2009, IEEE. p. 160-167.

[9] Cohen, M.B., Designing Test Suites for Software Interactions Testing. 2004, Designing Test Suites for Software Interactions Testing: Department of Computer Science.

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

[11] Stardom, J., Metaheuristics and the search for covering and packing arrays. 2001, Simon Fraser University.

[12] Shiba, T., T. Tsuchiya, and T. Kikuno. Using artificial life techniques to generate test cases for combinatorial testing. in Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International. 2004. IEEE.

[13] Kacker, R.N. and J.T. Jimenez, Tower of Covering Arrays. 2015.

[14] HOMAID, A.B., et al., Adapting the Elitism on the Greedy Algorithm for Variable Strength Combinatorial Test Cases Generation. IET Software, 2018.

[15] Homaid, A.A.B., et al., A Kidney Algorithm for Pairwise Test Suite Generation. Advanced Science Letters, 2018. 24(10): p. 7284-7289.

[16] Afzal, W., R. Torkar, and R. Feldt, A systematic review of search-based testing for non-functional system properties. Information and Software Technology, 2009. 51(6): p. 957-976.

[17] Bryce, R.C. and C.J. Colbourn. One-test-at-a-time heuristic search for interaction test suites. in Proceedings of the 9th annual conference on Genetic and evolutionary computation. 2007. ACM.

[18] Maity, S. and A. Nayak. Improved test generation algorithms for pair-wise testing. in Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on. 2005. IEEE.

[19] Yilmaz, C., M.B. Cohen, and A. Porter. Covering arrays for efficient fault characterization in complex configuration spaces. in ACM SIGSOFT Software Engineering Notes. 2004. ACM.

[20] Zamli, K.Z., M.H.M. Hassin, and B. Al-Kazemi. tReductSA–Test Redundancy Reduction Strategy Based on Simulated Annealing. in International Conference on Intelligent Software Methodologies, Tools, and Techniques. 2014. Springer.

[21] Cohen, M.B., C.J. Colbourn, and A.C. Ling, Constructing strength three covering arrays with augmented annealing. Discrete Mathematics, 2008. 308(13): p. 2709-2722.

[22] Sabharwal, S., et al., Construction of mixed covering arrays for pair-wise testing using probabilistic approach in genetic algorithm. Arabian Journal for Science and Engineering, 2016. 41(8): p. 2821-2835.

[23] Bansal, P., et al., Construction of variable strength covering array for combinatorial testing using a greedy approach to genetic algorithm. e-Informatica Software Engineering Journal, 2015. 9(1).

[24] Bansal, P., et al. An approach to test set generation for pair-wise testing using genetic algorithms. in International Symposium on Search Based Software Engineering. 2013. Springer.

[25] Flores, P. and Y. Cheon. PWiseGen: Generating test cases for pairwise testing using genetic algorithms. in Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on. 2011. IEEE.

[26] McCaffrey, J.D. An empirical study of pairwise test set generation using a genetic algorithm. in the 2010 Seventh International Conference on Information Technology: New Generations (ITNG). 2010. IEEE.

[27] Sthamer, H.-H., The automatic generation of software test data using genetic algorithms. 1995, University of Glamorgan.

[28] Wu, H., et al., A discrete particle swarm optimization for covering array generation. IEEE Transactions on Evolutionary Computation, 2015. 19(4): p. 575-591.

[29] Mahmoud, T. and B.S. Ahmed, An efficient strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software testing use. Expert Systems with Applications, 2015. 42(22): p. 8753-8765.

[30] 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.

[31] Zamli, K.Z., A.A. Al-Sewari, and M.H.M. Hassin, On Test Case Generation Satisfying the MC/DC Criterion. International Journal of Advances in Soft Computing & Its Applications, 2013. 5(3).

[32] AbdulRahman A. Alsewari, K.Z.Z., Interaction Test Data Generation Using Harmony Search Algorithm. 2011.

[33] Nasser, A.B., et al., A cuckoo search based pairwise strategy for combinatorial testing problem. Journal of Theoretical and Applied Information Technology, 2015. 82(1): p. 154.

[34] Nasser, A.B., A.R.A. Alsewari, and K.Z. Zamli. Tuning of cuckoo search based strategy For T-Way testing. in International Conference on Electrical and Electronic Engineering. 2015.

[35] Alsariera, Y.A., A. Nasser, and K.Z. Zamli, Benchmarking of Bat-inspired interaction testing strategy. International Journal of Computer Science and Information Engineering (IJCSIE), 2016. 7: p. 71-79.

[36] Alsariera, Y.A. and K.Z. Zamli, A bat-inspired strategy for t-way interaction testing. Advanced Science Letters, 2015. 21(7): p. 2281-2284.

[37] Alsariera, Y.A., M.A. Majid, and K.Z. Zamli, Adopting the bat-inspired algorithm for interaction testing, in The 8th edition of annual conference for software testing. 2015. p. 14.

[38] Alsariera, Y.A., M.A. Majid, and K.Z. Zamli, SPLBA: An interaction strategy for testing software product lines using the Bat-inspired algorithm, in 4th International Conference on Software Engineering and Computer Systems (ICSECS). 2015, IEEE. p. 148-153.

[39] Alsariera, Y.A., M.A. Majid, and K.Z. Zamli, A bat-inspired Strategy for Pairwise Testing. ARPN Journal of Engineering and Applied Sciences, 2015. 10: p. 8500-8506.

[40] Alsewari, A.A., et al., ABC Algorithm for Combinatorial Testing Problem. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 2017. 9(3-3): p. 85-88.

[41] Alazzawi, A.K., et al., Artificial Bee Colony Algorithm for Pairwise Test Generation. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 2017. 9(1-2): p. 103-108.

[42] Nasser, A.B., et al. Assessing optimization based strategies for t-way test suite generation: the case for flower-based strategy. in Control System, Computing and Engineering (ICCSCE), 2015 IEEE International Conference on. 2015. IEEE.

[43] Nasser, A.B., et al. Sequence and sequence-less T-way test suite generation strategy based on flower pollination algorithm. in Research and Development (SCOReD), 2015 IEEE Student Conference on. 2015. IEEE.

[44] Nasser, A., et al., Late acceptance hill climbing based strategy for addressing constraints within combinatorial test data generation. 2014.

[45] Nie, C. and H. Leung, A survey of combinatorial testing. ACM Computing Surveys (CSUR), 2011. 43(2): p. 11.

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

[47] Hartman, A., Software and hardware testing using combinatorial covering suites, in Graph theory, combinatorics and algorithms. 2005, Springer. p. 237-266.

[48] Czerwonka, J. Pairwise testing in the real world: Practical extensions to test-case scenarios. in Proceedings of 24th Pacific Northwest Software Quality Conference, Citeseer. 2006.

[49] Cohen, D.M., et al., The AETG system: An approach to testing based on combinatorial design. Software Engineering, IEEE Transactions on, 1997. 23(7): p. 437-444.

[50] Bryce, R.C. and C.J. Colbourn, A density ‐ based greedy algorithm for higher strength covering arrays. Software Testing, Verification and Reliability, 2009. 19(1): p. 37-53.

[51] Bryce, R.C. and C.J. Colbourn, The density algorithm for pairwise interaction testing. Software Testing, Verification and Reliability, 2007. 17(3): p. 159-182.

[52] Arshem, J., TVG. http://sourceforge.net/projects/tvg,, 2010.

[53] Tung, Y.-W. and W.S. Aldiwan. Automating test case generation for the new generation mission software system. in Aerospace Conference Proceedings, 2000 IEEE. 2000. IEEE.

[54] Yu, Y., S.P. Ng, and E.Y. Chan. Generating, selecting and prioritizing test cases from specifications with tool support. in Quality Software, 2003. Proceedings. Third International Conference on. 2003. IEEE.

[55] Lehmann, E. 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.

[56] Jenkins, Jenny. http://www.burtleburtle.net/bob/math/, 2003.

[57] Cohen, M.B., M.B. Dwyer, and J. Shi. Interaction testing of highly-configurable systems in the presence of constraints. in Proceedings of the 2007 international symposium on Software testing and analysis. 2007. ACM.

[58] Lei, Y., et al., IPOG: A general strategy for t-way software testing, in Engineering of Computer-Based Systems, 2007. ECBS'07. 14th Annual IEEE International Conference and Workshops on the. 2007, IEEE. p. 549-556.

[59] Forbes, M., et al., Refining the in-parameter-order strategy for constructing covering arrays. Journal of Research of the National Institute of Standards and Technology, 2008. 113(5): p. 287.

[60] Calvagna, A. and A. Gargantini. IPO-s: incremental generation of combinatorial interaction test data based on symmetries of covering arrays. in Software Testing, Verification and Validation Workshops, 2009. ICSTW'09. International Conference on. 2009. IEEE.

[61] Wang, Z., B. Xu, and C. Nie. Greedy heuristic algorithms to generate variable strength combinatorial test suite. in Quality Software, 2008. QSIC'08. The Eighth International Conference on. 2008. IEEE.

[62] Esfandyari, S. and V. Rafe, A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy. Information and Software Technology, 2018. 94: p. 165-185.

[63] Zamli, K.Z., B.Y. Alkazemi, and G. Kendall, A Tabu Search hyper-heuristic strategy for t-way test suite generation. Applied Soft Computing, 2016. 44: p. 57-74.

[64] Karaboga, D., An idea based on honey bee swarm for numerical optimization. 2005, Technical report-tr06, Erciyes university, engineering faculty, computer engineering department.

[65] Karaboga, D. and B. Akay, A comparative study of artificial bee colony algorithm. Applied mathematics and computation, 2009. 214(1): p. 108-132.

[66] Kuhn, R. ACTS download page. [cited 2018 2018]; Available from: http://csrc.nist.gov/groups/SNS/acts/download_tools.html

[67] Wu, H., et al. DPSO download page. [cited 2018 2018]; Available from: https://github.com/waynedd/DPSO.