Artificial Bee Colony Algorithm for Pairwise Test Generation

Ammar K. Alazzawi[1], Ameen A. Ba Homaid[1], Alaa A. Alomoush[1], AbdulRahman A. Alsewari[1,2]
[1]*Faculty of Computer Systems & Software Engineering, Universiti Malaysia Pahang, Kuantan, Pahang, Malaysia.*
[2]*IBM Centre of Excellence.*
ammarkareem91@gmail.com

***Abstract*—Our dependence on software applications has become dramatic in many activities of our daily life as they help to increase the efficiency of our tasks. These software applications have many sets of input values, parameters, software/hardware environments and system conditions, which need to be tested to ensure software reliability and quality. However, the whole comprehensive software testing is virtually not possible due to marketing pressure and resource constraints. In an attempt to solve this problem, there has been a development of a number of sampling and pairwise strategies in the literature. In this paper, we evaluated and proposed a pairwise strategy named Pairwise Artificial Bee Colony algorithm (PABC). According to the benchmarking results, the PABC strategies outdo some existing strategies to generate a test case in many of the system configurations taken into consideration. In a case where PABC is not at its optimal stage or its best performance, the experiments of a test case are effectively competitive. PABC progresses as a means to achieve the effective use of the artificial bee colony algorithm for pairwise testing reduction.**

***Index Terms*—Interaction Testing; Test Data Generation; T-way Testing; Software Testing; Natural Based Search Algorithms; Optimizations Problems.**

I. INTRODUCTION

Similar to any other engineering processes, software development is subjected to cost. Nowadays, software testing (as a process of the SDLC) consumes most of the time and cost spent on software development. This cost may decrease rapidly as testing time decreases. Most of the time, the software may be released without being tested sufficiently because of marketing pressure as well as the intention to save time and cut cost. However, releasing low-quality software products to the market is no longer acceptable because it may cause a loss of revenue or even a loss of life. Therefore, software tester should build high-quality test cases, which can detect the defects in the software without exceeding the required testing time. In this case, the test case minimization techniques take a great part in reducing the number of test case size without affecting their quality. Hence, the reduction of test cases, particularly in the configurable software systems is a primary issue.

Recently, configurable software systems have gained paramount usefulness in the market due to their capability to change the way a software behaves via configuration. The Traditional test techniques are essential for detecting and preventing defects, but it is not meant for eliminating defects due to the combinations of configuration and components input [1]. We consider that all the combinations configuration result in comprehensive testing, which is not possible due to resource constraints and time factor. Some test cases can be minimized if efficient test cases are designed to have the same effect as comprehensive testing [2].

In the past 20 years, the existing strategies of software testing to solve the problem have been developed [3]. Among these techniques, the combinatorial testing techniques are the most useful for designing test cases to solve this problem. These strategies help find and produce sets of tests to form a final test suite that helps in covering the needed combinations in compliance with the combination degree or strength. The degree begins from 2 ($t = 2$, where t represents the degree or strength of combinations).

We consider that all the sets that reduce to the minimum test suite are a difficult computational optimization issue [4] because finding the optimum set is an NP-hard issue (nondeterministic polynomial time) hard problem [5]. Therefore, looking for an optimal combination of test cases can be a challenging mission, and getting a unified strategy in order to produce an optimal outcome is challenging. There are two ways that can be taken to solve this issue effectively and to get a close-optimal result. The first direction utilizes computational algorithms using a mathematical arrangement, while the other uses nature-inspired algorithms [6].

Utilizing algorithms inspired by nature can produce extra effective outcome than the computational algorithms with a mathematical arrangement [6], [7]. Further, this method is more flexible than the other methods because it can create many combinatorial sets with various input levels and factors. Therefore, its result is more useful because most actual-world systems [8] have various input levels and factors.

One of the major problems in pairwise is in the generation of the best test case set (which is every pairwise interaction is covered by only one test case whenever possible) from a big probable test parameter numbers. Therefore, finding the ideal test cases is a complicated issue of NP. It means that any rise in the magnitude of the parameter causes the exponential rises in the estimated computational time and in the extent of the intricacy of the problem [9], [10]. Due to this, a lot of strategies (and their tool execution) have been structured in literature.

To address the problems above and as a completion of the existing work, we have proposed the use of artificial bee colony algorithm for pairwise strategy called pairwise artificial bee colony algorithm (PABC) strategy.

This paper is organized as follows. Section I presents the background for software testing and pairwise technique.

Section II covers an array of definition, while in Section III, a model is illustrated using a display tab. Section IV reviews the existing combinatorial test data generation strategies. Section V illustrates the design and implementation of a PABC, including the algorithms. Section VI discusses the comparison results of the different experiments to evaluate the performance of PABC. Finally, in section VII we present our conclusions and the suggestions for future works.

II. Covering Array Definition

T-way testing interaction test suite can be abstracted via the covering array (CA) notations. Typically, the CA has 4 factors; t, N, v, and p, (CA (t, N, vp). The symbols t, p, and v depict the number of interaction strength, parameters, and values for the CA, correspondingly. For instance, CA (9, 2, 34) denotes a test suite entailing of 9x4 ranges (the rows denote the test case size (N), and the column denotes the parameter (p)). The sizes of the test suite encompass 2-way interaction for a system with 4 three-value parameters.

Additionally, to CA there is MCA (mixed covering array) with 3 parameters; t, C (Configuration) and N (i.e., MCA (N, t, C)). In addition to t, and N that conveys a similar meaning as in CA, the MCA employs a new symbol, C which is consistent with the previously given representations. C denotes the values and parameters of every of the configurations in the given formats: v^1p^1, v2p2… v^np^n depicting that there are p1 parameters with v1 values, p^2 parameters with v2 values till V^np^n. For example, MCA (1265, 4, 10^2 41 3^2 2^7) shows the 1265 test size that covers 4-way interaction. The MCA configuration requires 12 parameters, which are 2 ten-value parameters, 1 four-value parameter, 2 three-value parameters, and 7 two-value parameters.

III. Problem Statement

In this section, we present a short definition to explain the connotation of t-way interaction testing. Consider a t-way testing as a more effective technique to create the most minimum test suite used to detect the mistakes of interaction. The main concept of using t-way testing is to show that not all parameters result in every parameter's error.

Overall, every system consists of a number of factors, which are called parameters with their value (that interact together). To clarify the conception of t-way interaction testing, we consider the display tab of a file as a simple example for the basis of our problem as shown in Figure 1.

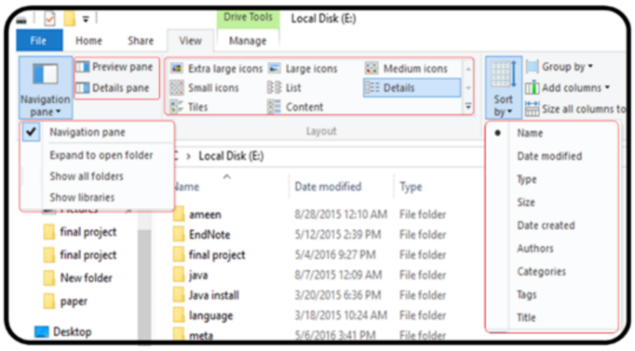

Figure 1: View of the folder

Figure 1 shows a screenshot of a display's tab for the file. The display's tab consists of five groups of features that have one or more variable or parameters: the Navigation pane group, preview pane and details pane group, layout group and sorted by group. The display's tab of file provides simple wide levels and factors (i.e., called parameters and values).

The display's tab consists of four parameters: one 4-value parameters (i.e. Navigation pane), two 2-value parameters (i.e. preview pane), one 8-value parameters (i.e. layout group) and one 9-value parameters (i.e. sort by). ◌At all, it has four system configuration. These system configurations (SC) include variable values except the preview pane parameter and detail's pane parameter, which include "select" or "unselect." In cases of other parameters, they include "select" or skipped "unselect." In case of "select" or "unselect", we represent the value of the parameters as (on) and (off) respectively.

The system configurations (SC) for the display tab are explained in Figure 1. Based on Figure 1, we assume t = 2^{11}. The covering array is represented as MCA (N, 2, 4^1 2^2 8^1 9^1). The total exhaustive combinations are ($4^1 \times 2^2 \times 8^1 \times 9^1 = 1152$) test cases. These are virtually ineffective, if they were done manually. If we assume to analyze a test case that requires five minutes, it takes 96 hours to examine only the display tab completely, which is probably not practical, according to the testing standards.

Pairwise testing is a simple technique space by generating a minimum test case, where the need for interaction strength of degree t is covered at least only once (where t indicates to the strength of degree). The use of pairwise testing (t-way testing) in our example is as shown in Table 1. Only 72 test cases can cover every pairs of parameters value input as minimum one time. Here, it lowers the number of test cases from 1152 to 72. Table 1 shows the results of our example with 10 times running, which includes minimum test case, average size, time, average time and best time size.

Table 1
Result of Display Tab (MCA (N, 2, 4^1 2^2 8^1 9^1)).

t	Best size	Avg. size	Best time	Avg. time	Best Size times
2	72	72.4	20.536999940	21.09489994	6

IV. Related Work

Overall the existing strategies for pairwise technique can be classified into three groups, depending on the prevailing approaches [12]:

A. *Algebraic construction category*

Here, the strategy for the construction of test sets is by using the mathematical function or pre-defined rules [12]. Therefore, the computations involved in algebraic approaches are typically lightweight, and in some cases, algebraic approaches can generate the most optimal test sets. However, the applicability of algebraic approaches is often restricted to small configurations [12], [13]. OA (Orthogonal arrays) [14], MOA (Mathematics of Arrays) [15] TConfig [16] are great examples of the strategies that depend on the algebraic approach.

B. *Greedy algorithm category*

The strategies in this category are mostly depending on the creation of every of the pair combinations. Unlike the algebraic approaches that depend on every pair of combinations, the computational approaches explore combinations space to produce the test cases needed until all of the pairs have been covered. In this manner, this category based strategies may typically be appropriate in large system configurations. However, in the case where the number of pairs to be considered is significantly large, then implementing greedy algorithm based approach can become very costly as a result of the necessity enumeration from all the combination space. An example of the strategies that employs this approach includes an AETG [17], [18], its variation mAETG [19], IPO [20], PICT [21], Jenny [22], TVG [23], IPOG [12], all pairs [24], CTE_XL [25], IRPS [26], and G2Way [27].

AETG [17], [18] and its variant mAETG [11] use a greedy random search algorithm depending on a 2-way interaction pairing to get the final test suite. Therefore, the created test case is not naturally deterministic. Regarding the PICT (Doug, & Keith 2006), it generates all the determined interaction first, and then randomly selects their corresponding interaction combinations to form the test cases as part of the complete test suite.

The IPO strategy [20] constructs an all-pairs test set for the initial 2 parameters. The IPO strategy that encompasses the test set covering the initial 3 parameters, is then in continual: It encompasses the test set until it creates all pair test set for the whole parameters. Apart from being deterministic, encompassing a parameter at a time lets the IPO strategy to attain a lesser rate of intricacy than the AETG. Lately, this strategy has been protracted to handle the advanced interaction strength in the improvement of the IPOG [12].

Test data were generated in some phases by Jenny [22]. Initially, Jenny produces test data to cover the whole one-way interaction. The initial phase test data will then be extended by Jenny so as to greedily encompass that of two-way interactions. Optionally, this method can be a continual process till the nth number of way interactions as stated by the user.

All pairs strategy ([24], CTE_XL [25] and TVG [23] share the same property as much as generating deterministic test cases is concerned though very few things are known about the real algorithms used due to limited availability of references.

The more contemporary strategies centered on the computational approaches are G2Way [27], IRPS [26]. IRPS focuses on effectual data structure for searching and storing pairs, and it is deterministic in nature. In this way, IRPS provides a comparatively fast effecting time when comparison with other strategies is made. G2Way adopts a backtracking algorithm to combine pairs to produce a pairwise test suite. G2Way also backs automated implementation of the produced test suite unlike other strategies that does not.

C. *Natural Search based category*

Regarding the implementation of NS based algorithm, much current work has started coming up to include particle swarm optimization for pairwise test generation (PSO) [28], pairwise harmony search strategy (PHSS) [29]-[32], genetic algorithm (GA) [33], ant colony algorithm (ACA) [33], and simulated annealing (SA) [34]. In GA, the test case creation process always begins with random test data (referred to as chromosomes later). The chromosomes will undergo through a sequence of mutation progressions till certain stopping criteria are met. The better chromosomes will be chosen as an ultimate test suite. Regarding ACA, test case creation process is simulating the colonies of ants that move from one spot to the other (representing the parameter) to get food (which represent the end of test case) through several paths (correspond to values for every parameter). The best route (gotten depending on the amount of the pheromone left by the colonies of ants) depicts the greatest value for a test case.

In a nut shell, SA adopts a probability based transformation equation alongside with a greedy binary exploration algorithm to get the best test case iteratively to encompass all the needed (pairwise) interfaces from a random selection space. Similarly, PSO, a PSO based strategy, iteratively executes global and local searches to get the candidate result to be added to a definitive suite till the whole pairwise interactions are covered. HSS, adopts the harmonic selection between the instruments.

V. ABC Algorithm

The ABC algorithm is designed to emulate the foraging behavior of a honey bee colony. A typical honey bee swarm consists of 3 essential segments: unemployed foragers /employed foragers (bees) /food source. Employed foragers are the bees that are employed at, and presently exploiting, a particular source of food. These bees convey data relating to the profitability direction and distance of the food source and also connect the data with every single honeybee in the hive. The unemployed honey bees are categorized as either a scout honeybee or an onlooker bee. The later strives to get a source for food with the use of the data given by the employed honey bee, while the latter randomly searches the surroundings to locate a new source of food (better) [35]. Presumably, an employed bee whose source food is depleted becomes a scout bee and begins to look out for another new source of food. Moreover, it assumes the aggregate number of the employed honey bees has to be the same as the number of sources of food. Imaginable, the position of a source for food depicts a probable test cases out to the optimization issue, though the quantity of a source of food relates to the fitness (quality) of the associated test cases.

Primarily, the ABC produces a randomly distributed population of SN test cases (positions of food source) in the exploration space, where SN represents the onlooker bees size or employed bees. Supposing the number of optimization parameters is D, then each of the test cases xi (i = 1, 2... SN) i basically will be a D-dimensional vector. Every result produced here can be attained from the Equation 1 [35];

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

Here, x_{min} and x_{max} depict both the upper and lower boundary parameters for the test cases x_i, while in dimension j (j=1, 2… D), and Rand [0, 1] is a scaling factor representing a random integer between [0,1]. The positions of the food source (D-dimensional results) produced in the initial step (C=0) are liable to repeating cycles C= (1, 2…, MCN), until a termination criterion is satisfied. Both the local and the global probable selection/search are implemented in a single cycle ABC. Each cycle comprises a number of responsibilities executed by various types of honey bee. These processes are principally independent, which can be elucidated in a distinct way as shown below, for more clarification of the ABC methodology:

A. *Employed Bee phase*

After the employed bee has been allocated to their sources of food, these honey bees assess the capability of their test cases s (sources) and converse the data with the onlooker honey bees. In addition, every of the employed honey bee produces a candidate food position (test cases) by perturbing the old source of food (test cases) if (x_{ij}) in its memory, using Equation 2 [35]:

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

Here, j∈{1,2,...,D} and k ∈{1,2,..., SN} (k ≠ i) are randomly selected indexes, and Rand [-1,+1] is a random number between [-1,1], which works as a scaling factor. It is obvious that as the optimal result in the search space is approached, this perturbation on result gets decreased. The capability of the perturbed (new) result will also be assessed by the employed bee, and in case when better fitness values are found, the new test cases replaces the old one in the memory of that employed bee (a greedy-selection scheme).

B. *Onlooker Bee phase*

The main duty of an onlooker bee is to choose a test cases (source of food), based on the possibility quantity associated with the source of food, Pi, which is evaluated by Equation 3 [36]:

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

where, fit denotes the fitness value of a given test cases, and the subscript index depicts the test cases number. This probable choice is affected by relating P_i against a randomly chosen number between [0, 1]. The selection is approved if the generated random number is lesser or equivalent to P_i , if otherwise it will be rejected. The duty of the onlooker honey bee to a specified test cases will be approved if the equivalent probable selection is sanctioned. Normally, in the minimization problems, the fitness value of test cases s is calculated by Equation 4:

$$fitness_i = \begin{cases} \dfrac{1}{1+f_i}, & if\, f_{i \geq 0} \\ 1 + |f_i|, & if f_{i<0} \end{cases} \quad (4)$$

where, f_i is the value of the objective function for test cases. After the selection of a food source (test cases) with a P_i possibility, the onlooker honey bee will select a new test cases (source of food) in the area of the preceding one in her memory, using Equation 2. In case the new test cases (food source) has a better fitness value, then an onlooker honey bee will update the new test cases (food source) in her mind, and forgets the old one, similar to the case with the employed bees.

C. *Scout Bee phase*

The duty of scout bees is to randomly explore the entire search space to get an improved (new) result to the overall optimization problem. Unlike the situation with onlooker /employed honey bees (where they are bound to create trial result round an old result), the scout honey bees are not bounded in this sense. The scout bees draw their samples from a wide set of D-dimensional vectors, so far it is inside the boundaries of the search space. In ABC, if a (non-global) test case cannot be developed further after a pre-determined number of cycles, then the test cases will be neglected, and the employed Bee allocated to that exact position will transform to a scout bee with essentially scout-type functionality. The value of this pre-determined number of cycles, which is termed the limit, will therefore be an important control parameter in the algorithm. In practice, the limit is estimated via Equation 5:

$$limit = c * n_e * D \quad (5)$$

where, n_e is the number of unemployed bees, and where c is a coefficient constant with an acclaimed value of 1 or 0.5 [17], [18]. However, one scout bee must exist during the implementation of ABC. The scout type processes give an excellent ability to the ABC process in getting the paramount global result, by creating stochastic inquiry in the whole D-dimensional area. This is to say that scout bees will independently search for a global optimal result, while all other types of bee (onlooker /employed) are concurrently scrutinizing their confined candidate test cases s for the overall best. For this reason, the probability of being ensnared in local optimum will never be appropriate to ABC.

VI. BENCHMARKING RESULTS

This section employed prevailing relative experimentations, which are stated in [26], [27], [29], [33]. So as to standardize the PABC strategy alongside the existing approach, we split our comparison into 2 parts. In the initial fragment, a system configuration with ten V-valued parameters were selected, where V varies (from three to ten) also a system configuration with P 2-valued parameters were selected, where P varies (from three to fifteen). The goal is to explore how PABC acts as regards changing P and V. For the second fragment, a number of system configurations into 11 groups to compare the performance of PABC alongside other strategies. The configurations are shown below:

S1 = CA (N, 2, 3^3)
S2 = CA (N, 2, 3^4)
S3 = CA (N, 2, 3^{13})
S4 = CA (N, 2, 10^{10})
S5 = CA (N, 2, 15^{10})
S6 = CA (N, 2, 10^{20})
S7 = CA (N, 2, 5^{10})
S8 = MCA (N, 2, $5^1\, 3^8\, 2^2$)
S9 = MCA (N, 2, $6^1\, 5^1\, 4^6\, 3^8\, 2^3$)
S10 = MCA (N, 2, $7^1\, 6^1\, 5^1\, 4^6\, 3^8\, 2^3$)
S11 = MCA (N, 2, $10^1\, 9^1\, 8^1\, 7^1\, 6^1\, 5^1\, 4^1\, 3^1\, 2^1$)

The shaded cells with asterisk (*) in Table 2 to 4 show the minimum generated size (test suite) for every strategy, and the shaded cells without an asterisk generated competitive sizes with other strategies. The marked cells by not available (NA) indicate that the results for these strategies are not reported in their publications. According to Table 2, it is obvious that PABC generates the most optimal test case in only one when V = 3, unlike to the PHSS is not affected by the number of value or parameters, where PHSS outperforms with all other strategies except in case when v = 8.

Regarding to Table 3, PABC, PHSS and PSO produce the smallest solutions in most of the cases. In Table 4, PHSS, IRPS, Jenny, PABC, and PPST produce the optimal test suite for S1. PABC, PHSS, ACA, GA, SA, PSO and IRPS produce the least limit for S2. The AETG outpaces all strategies in S3. PHSS outdo all strategies in S4 excluding IRPS. While in S5, PHSS produces an acceptable result with regard to TVG, AllPairs, TConfig, G2Way, IPO, and IPOG while IRPS creates the optimal result. The SA produces the optimal result in S6 case. PHSS produces the optimal test suite in magnitude than all approaches in S7. In S7, PHSS produces an acceptable result with regard to TVG, PICT, CTX-XL, TConfig, AllPairs, IPO, IPOG and G2Way. Taking into consideration the size of the test suite for S8, GA&SA outpaces other strategies. In the case of the S9, Jenny generates the greatest size. As for S10, ACA and GA outpace all other strategies. Lastly, in the case of S10, IPOG outpace other strategies.

After considering all of the outcomes, two understated conclusions may be deduced. Primarily, no solitary strategy can assert control. Additionally, Natural search-based strategies tend to outpace other strategies. PABC, PPSTG, ACA, SA, PHSS and GA mostly provide competitive outcomes about other computational centered approaches. This outcome is anticipated as the aforementioned existing strategies that have their base from optimization processes.

Table 2
CA (N; 2, V10), V is variable from 3 to 10

V	PHSS	jenny	IPOG	TConfig	CTE_XL	PICT	TVG	PABC
	b	*b*	*b*	*B*	*b*	*b*	*b*	*B*
3	17	19	20	17	18	18	18	***16**
4	***28**	30	31	31	33	31	33	30
5	***43**	45	50	48	50	47	50	46
6	***60**	62	68	64	71	66	72	66
7	**79***	83	90	85	97	88	98	90
8	**105**	**104***	117	114	125	112	124	118
9	**127***	129	142	139	161	139	152	149
10	**155***	157	176	170	192	170	189	184

Table 3
CA (N; 2, 2P), P is variable from 3 to 15

P	PHSS	jenny	IPOG	TConfig	CTE_XL	PICT	TVG	PABC
	b	*b*	*b*	*b*	*b*	*b*	*b*	*B*
3	**4**	5	**4**	**4**	6	**4**	**4**	**4**
4	6	6	6	6	6	**5**	6	**5**
5	**6**	7	**6**	**6**	**6**	7	**6**	**6**
6	7	8	8	7	8	**6**	**6**	7
7	**7**	8	8	9	8	**7**	8	**7**
8	8	8	8	9	8	***7**	8	8
9	**8**	**8**	**8**	9	9	9	**8**	**8**
10	**8**	10	10	9	9	9	9	**8**
11	**8**	9	10	**9**	10	9	9	9
12	**9**	10	10	**9**	10	**9**	10	**9**
13	**9**	10	10	**9**	10	**9**	10	**9**
14	10	10	10	**9**	10	10	10	**9**
15	10	10	10	**9**	10	10	10	**9**

Table 4
Comparison with other existing strategies in terms of generated test suite for 11 system configurations

S	ACA	CTE-XL	TConfig	AllPairs	Jenny	TVG	PICT	AETG	mATEG	SA	GA	IPO	IPOG	IRPS	G2Way	PSO	PHSS	PABC
S1	NA	10	10	10	**9**	11	10	NA	NA	NA	NA	NA	11	**9**	10	**9**	**9**	**9**
S2	**9**	10	10	10	13	12	13	**9**	11	9*	**9**	9	12	**9**	10	**9**	**9**	**9**
S3	17	21	20	22	20	20	20	**15***	17	16	17	17	20	17	19	18	18	20
S4	159	192	170	177	157	189	170	NA	NA	NA	157	169	176	**149***	160	156	155	184
S5	NA	NA	NA	390	336	473	NA	NA	NA	NA	NA	361	373	**321***	343	NA	341	427
S6	225	NA	NA	230	NA	NA	NA	180	198	**183***	227	212	NA	210	200	213	224	283
S7	NA	50	48	49	45	50	47	NA	NA	NA	NA	47	50	45	46	45	**43***	46
S8	16	21	22	21	41	23	21	19	20	**15**	**15**	NA	19	17	23	17	20	20
S9	32	39	33	NA	**31***	41	38	34	35	NA	33	NA	36	NA	NA	35	39	39
S10	**42**	53	79	NA	51	52	46	45	44	NA	**42**	NA	44	NA	NA	43	48	47
S11	NA	102	92	NA	98	100	101	NA	NA	NA	NA	NA	**91***	NA	NA	97	95	97

VII. CONCLUSION

In this paper, we have set a new strategy termed PABC centered on ABC algorithm for pairwise test case generation. The investigation results show that our PABC's works well and overcomes other existing strategies in some cases measured. As part of the forthcoming work, we look further to develop the execution of PABC. Presently, we work to enhance great interaction strength and tackle the problem of constraints and seeding.

ACKNOWLEDGMENT

This research is partially funded by UMP RDU130366 Short Term Grant: Development of a Pairwise Testing Tool with Constraint and Seeding Support Based on and Optimization Algorithm, UMP RDU150369: A new Hybrid Variable Interaction Strength Test Data Generation Strategy Based on Harmony Search Algorithm and Cuckoo Search Algorithm, UMP RDU Grant: Modified Greedy Algorithm Strategy for Combinatorial Testing Problem with Constraints Supports, and FRGS RDU130119 Grant: Input Output Relations Harmony Search T-way Testing Strategy.

REFERENCES

[1] M. B. Cohen, 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) 129-139.

[2] Y. A. Alsariera, M. A. Majid, and K. Z. Zamli, "A Bat-inspired Strategy for Pairwise Testing," ARPN Journal of Engineering and Applied Sciences, vol. 10, (2015) 8500-8506.

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

[4] V. V. Kuliamin and A. Petukhov, "A survey of methods for constructing covering arrays," Programming and Computer Software, vol. 37, (2011) 121-146.

[5] A. Ouaarab, B. Ahiod, and X.-S. Yang, "Discrete cuckoo search algorithm for the travelling salesman problem," Neural Computing and Applications, vol. 24, (2014) 1659-1669.

[6] C. Nie and H. Leung, "A survey of combinatorial testing," ACM Computing Surveys (CSUR), vol. 43, (2011) 11.

[7] P. McMinn, "Search-based software test data generation: A survey," Software Testing Verification and Reliability, vol. 14, (2014) 105-156.

[8] Y. A. Alsariera and K. Z. Zamli, "A Bat-inspired strategy for t-way interaction testing," Advanced Science Letters, vol. 21, (2015) 2281-2284.
[9] P. Danziger, E. Mendelsohn, L. Moura, and B. Stevens, "Covering arrays avoiding forbidden edges," Theoretical Computer Science, vol. 410, (2009) 5403-5414.
[10] X. Yuan, M. B. Cohen, and A. M. Memon, "GUI interaction testing: Incorporating event context," Software Engineering, IEEE Transactions on, vol. 37, (2011) 559-574.
[11] M. B. Cohen, "Designing test suites for software interaction testing," AUCKLAND UNIV(NEW ZEALAND), (2004).
[12] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: A general strategy for t-way software testing," in Engineering of Computer-Based Systems, ECBS'07. 14th Annual IEEE International Conference and Workshops, (2007) 549-556.
[13] J. Yan and J. Zhang, "A backtracking search tool for constructing combinatorial test suites," Journal of Systems and Software, vol. 81, (2008) 1681-1693.
[14] A. S. Hedayat, N. J. A. Sloane, and J. Stufken, Orthogonal arrays: theory and applications: Springer Science & Business Media, (1999).
[15] R. Mandl, "Orthogonal Latin squares: an application of experiment design to compiler testing," Communications of the ACM, vol. 28, (1985) 1054-1058.
[16] W. AW, (2002) "TConfig,".
[17] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatorial design," IEEE Transactions on Software Engineering, vol. 23, (1997) 437-444.
[18] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton, "The combinatorial design approach to automatic test generation," IEEE software, vol. 13, (1996) 83-88.
[19] C. J. Colbourn, M. B. Cohen, and R. Turban, "A deterministic density algorithm for pairwise interaction coverage," in IASTED Conf. on Software Engineering, (2004) 345-352.
[20] 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) 254-261.
[21] D. H. Keith, (2006) "PICT.,".
[22] Jenkins, (2003) "Test Tool,".
[23] J. Arshem, (2010) "TVG ", ed.
[24] B. J, (2001) "Allpairs Test Case Generation Tool."
[25] 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 (EuroSTAR2000), (2000); Kopenhagen, Denmark.
[26] M. I. Younis, K. Z. Zamli, and N. A. M. Isa, "IRPS–an efficient test data generation strategy for pairwise testing," in Knowledge-Based Intelligent Information and Engineering Systems, (2008) 493-500.
[27] M. F. Klaib, K. Z. Zamli, N. A. M. Isa, M. I. Younis, and R. Abdullah, "G2Way a backtracking strategy for pairwise test data generation," in Software Engineering Conference, 2008. APSEC'08. 15th Asia-Pacific, (2008) 463-470.
[28] X. Chen, Q. Gu, J. Qi, and D. Chen, "Applying particle swarm optimization to pairwise testing," in Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual, (2010) 107-116.
[29] A. R. A. Alsewari and K. Z. Zamli, "A harmony search based pairwise sampling strategy for combinatorial testing," International Journal of the Physical Sciences, vol. 7, (2012) 1062-1072.
[30] A. R. A. Alsewari, M. I. Younis, and K. Z. Zamli, "Generation of Pairwise Test Sets using a Harmony Search Algorithm," COMPUTER SCIENCE LETTERS, vol. 3, (2011).
[31] 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, (2012) 553-568.
[32] A. R. A. Alsewari, N. Khamis, and K. Z. Zamli, "Greedy interaction elements coverage analysis for AI-based t-way strategies," Malaysian Journal of Computer Science, vol. 26, (2013) 23-33.
[33] 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, COMPSAC2004, (2004) 72-77.
[34] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn, "Constructing test suites for interaction testing," in Software Engineering, 2003. Proceedings. 25th International Conference on, (2003) 38-48.
[35] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," Applied mathematics and computation, vol. 214, (2009) 108-132.
[36] D. Karaboga and C. Ozturk, "A novel clustering approach: Artificial Bee Colony (ABC) algorithm," Applied soft computing, vol. 11, (2011) 652-657.