Test Data Generation for Event Driven System Using Bees Algorithm

Mohd Hazli Mohamed Zabil[1] and Kamal Z. Zamli[2]

[1]*College of Information Technology*
Universiti Tenaga Nasional, Putrajaya Campus,
Jalan IKRAM-UNITEN,
43000 Kajang, Selangor, Malaysia.

[2]*Faculty of Computer Systems and Software Engineering,*
Universiti Malaysia Pahang
26300 Gambang, Pahang, Malaysia.

Abstract- For an event driven system, the order of the event sequence should also be tested to detect failure in any possible sequences of the event. In many real time or reactive system, some faults do occur as a result interactions of some particular order of the inputs or events. In some other systems, sequence of inputs produce significant results to how such system process the inputs and produce the output. For these types of systems, fault might be triggered from a particular order of the input sequence, entered or given to the system. . In this paper we discuss and proposed a new strategy for generating test data for event-driven system using a bio inspired artificial intelligent, namely Bees Algorithm (BA). We discussed the implementation of BA and benchmark it with the existing approaches.

Keywords: Interaction testing; Sequence-based Interaction testing; sequence covering array; Bees Algorithm

I. INTRODUCTION

One of the important objectives of software testing is to find as many faults as possible in a system under test (SUT). As there are many classes of systems, in order to achieve the objective, different types and approaches of testing might need to be perform (e.g. regression testing, performance testing, compatibility testing and interaction/combinatorial testing). For an event-driven system, faults may be triggered from a combination of events triggered during a process. To detect such faults, combinatorial interaction testing (CIT) can be performed against the SUT. This is to ensure that combination of any events will not flag any errors. In the literature, many CIT strategy have been developed for the past 20 years (e.g. Jenny[1], IPOG[2], MC-IPOG[3], AETG[4] and TConfig[5], ACS[6], PICT[7], PSO[8], HSS[9], ParaOrder[10], Density[11], TVG[12] and ITTDG[13]). Although these strategies are able to detect faults due to combination or interaction between events, the sequence of events occur are not being considered. This would risk the SUT to faults due to sequence of events. In an event-driven system, event can occur in many sequences. In order to detect the fault due to different sequences, we need to test all possible sequences of each event for the SUT. However, testing all possible sequences, even for a small system is inefficient and yet affordable due to resource constraints (i.e. time, budget and human resource).

Recently, Kuhn et.al[14] and Esra et.al[15] have proposed a new approaches to generate test data for testing event-driven system. Their work, focus on systems with distinct number of event and each event occurs only once, since event repetition is not always the case in an event-driven system. Kuhn has been using computational greedy approach while Esra is proposing a rule-based approach using Answer Set programming (ASP). Both approaches have its strength and limitations; hence we are looking into improving the limitations of the two approaches. The advantages and limitations of the two mentioned approaches are discussed in the next following section. In recent development, researchers have been adopting artificial intelligence (AI) to solve combinatorial optimization problems. In the emerging research area of software testing, researchers have been proposing the use of AI to generating efficient test data for interaction testing (e.g. GA[16], ACS[6], PSO[8] and HSS[9]). From the published results, in general, the AI-based strategies produce smaller test suite size compared to the computational strategies especially when it comes to higher number of parameters (events). Motivated by this works, we propose a new approach using bio-inspired AI algorithm (namely Bees Algorithm, BA) to optimize the generation of test data for sequence of event-driven system as mentioned above.

For the purpose of presentation, this paper is organized as follows. Section 2 discusses the preliminaries; Section 3 highlights related works; Section 4 discusses the problem definition model; Section 5 design and implementation of BA strategy; Section 6 elaborates the results and benchmarking against existing strategies; Section 7 presents the case study and Section 8 provides the conclusion.

II. PRELIMINARIES

Sequence Covering Array (SCA) introduced by Kuhn[14] is comparable to Covering Array[17], a combinatorial object that has been improved from Latin Square[18] and

Orthogonal Array[19]. Covering array is used in generating test cases for combinatorial testing. In each row of covering array, each combination of *t*-way parameter is covered at least once for fixed *t* strength. In order to include sequence in covering array algorithm, constraints need to be introduced to make sure each row contains a particular value only once, which might be highly inefficient[14]. Based on combinatorial method of covering array, Kuhn proposed sequence covering array (SCA). SCA functions like a covering array, but also consider the sequence of each *t*-way events to be covered in each row only once. SCA is designed specifically to address system with unrepeated event sequence. SCA is written as SCA(N, *S*, *t*) where N is the size of the test suite, *t* is the strength and *S* is the number of event sequence [14], [15]. Each row of N contains a permutation of *s* symbol of set S and at least one *t*-way sequence is covered in every N rows. SCA is optimal if the number of rows (N) is minimum. For example, take a system with 5 events (A,B,C,D and E). To test exhaustively all 5 sequence (i.e. t=5), we would have 5! = 120 test cases. Since exhaustive testing is inefficient and unaffordable, we could relax the strength to 3 (i.e. t=3). For t=3, there are 60 possible 3-way sequence as in Figure 1.

{ABC}{ABD}{ABE}{ACB}{ACD}{ACE}{ADB}{ADC}
{ADE}{AEB}{AEC}{AED}{BAC}{BAD}{BAE}{BCA}
{BCD}{BCE}{BDA}{BDC}{BDE}{BEA}{BEC}{BED}
{CAB}{CAD}{CAE}{CBA}{CBD}{CBE}{CDA}{CDB}
{CDE}{CEA}{CEB}{CED}{DAB}{DAC}{DAE}{DBA}
{DBC}{DBE}{DCA}{DCB}{DCE}{DEA}{DEB}{DEC}

Figure 1. Exhaustive 3-way sequence

For this example, based on [14], the first random test case (e.g. A,B,C,D,E), would cover up to 10 uncovered sequence (i.e. ABC, ABD, ABE, ACD, ACE, ADE, BCD, BCE, BDE and CDE). Hence, we could perform some optimization using greedy heuristic, to find a test case that cover the most, so fewer test cases needed to cover all 3-way sequence. After optimization, from 60 3-way sequence, we only need 8 test cases to cover all 3-way sequence i.e. SCA(8, 5,3). It has been proven that finding optimal covering arrays by searching for a test case that covers the most is NP-complete [4][20][21][22]. Since SCA is also using combinatorial method, finding the next test case that cover the most uncovered set is also NP-Complete. One popular approach to solve this covering set problem is by using greedy approach. Although greedy algorithm cannot guarantee the most optimal result, a good greedy technique could produce a close approximation to the most optimal result [23].

III. RELATED WORKS

In this section we discuss existing approaches in generating SCA, namely *t-seq* by Kuhn et.al[14] and ASP by Esra et,al[15]. Kuhn proposed a greedy algorithm to generate SCA called t-seq. In general, *t-seq* is a greedy algorithm. A number of candidate test cases are generated and each candidate test case is checked against the uncovered sequence. The test case which covers the most sequences will be selected. If a constraint is defined, the test candidate with unwanted sequence will not be selected. If constraint is not defined, a heuristic step is added where the selected test case will be reversed to get the same number of covered sequence. The reversal step is optional and can be turned off to allow all test candidate are generated randomly. The process iterates until all t-way sequences are covered. *t-seq* generates good results. Since *t-seq* is a computational greedy algorithm, the execution time is fast and the number of events is fairly scalable. Another approach of generating SCA has been proposed by Esra et.al using a declarative language, Answer Set Programming (ASP). In this approach, a finite set of rules known as answer sets are designed. The rules consist of symbols (events) definitions and conditions (i.e. how symbols should be arranged). Rules for checking if a particular sequence is covered also defined in the answer set. Once the ASP encoding is completed, the ASP solver will generate SCA based on the rules. Constraint rules can also be added to in the answer set rules.

t-seq is a simple and straight forward algorithm to generate SCA. Since *t-seq* is a computational approach, its advantage is in term of fast execution time and scalability in term of number of events. Although *t-seq* produces good results in term of size of SCA, ASP produces more optimal SCA size. In term of constraint support, *t-seq* provides simple constraint support and manual intervention need to be performed in order to achieve the constraint. Since ASP is a rule based approach, it has the advantage to include more complex constraints in the rules. One drawback of ASP is scalability. The rules need to be changed for different number of events. Since ASP is not designed to be a generic SCA generator, the SCA generated by ASP are limited only up to 40 events. This is due to the huge number of 4 sequences that need to be represented in the answer set. In term of sequence strength, as far as published results is concern, both t-seq and ASP are limited only up to strength 4 (i.e. t=4).

Motivated by the limitation of both *t-seq* and ASP, we are proposing a bio-inspired AI-based approach in generating SCA. As mentioned earlier, many published works have shown that AI-based algorithms have been used in solving optimization problems and produce good results compared to computational approached. Moving from here, we adopt a relatively new AI-based algorithm, namely Bees Algorithm (BA) for generating SCA.

IV. PROBLEM DEFINITION MODEL

To appreciate the application of SCA, consider a house alarm system with 4 input sensors as in Table 2. When the alarm system is activated, the system will monitor these four sensors and produces an alarm messages when the system senses any unusual activities (e.g. in the presence of intruder).

TABLE 1 House Alarm System

Sensor	Function
A	Active when main door is opened
B	Active when any window is opened
C	Active when any window is broken
D	Detect any movement inside the house

In this scenario, the alarm system receives inputs from the sensors in random sequences. Main door may be opened first and then followed by any window opened or vice versa. Ideally, the house alarm system must be tested in all possible sequences so that no false alarm will be triggered due to sequencing faults. In a mass factory production of the alarm system, testing exhaustively is not feasible due to time-to-market, resources, and cost constraints. Even if the system has been tested during its manufacturing process, the system also needs to be tested after installation at the particular premise.

To test the system after installation on site, the testing procedure requires manual and physical intervention to trigger the sensors. This would be time consuming; thus, efficient test data are needed to reduce testing time. Should we consider to test all possible sequences (4 input sequences give 4!=24 tests) it may not be practical. If one test requires 5 minutes, we need 120 minutes or 2 hours to perform the testing at the installation site. This is obviously inefficient and increase cost related to installation. For this reason, there is a need for a sampling mechanism that can help to find the subsets of effective sequences for testing the scenario. Using SCA, testing 2-way sequence will give us SCA(2, 4,2), that is SCA of 4 events with strength of 2 and size of 2. Using a simple greedy approach testing 3-way sequence will give us 6 test cases.

V. BA STRATEGY

A. Overview of Bees Algorithm

Bees Algorithm (BA) has been proposed by Pham [24] and is a relatively new nature inspired, population based algorithm. BA is inspired based on the food foraging behavior of honey bee colony. In a honey bee colony, a group of scout bees are sent out randomly to find flower patches. In nature, honey bee can travel more than 10 kilometers in multiple directions from its hive. When return to the hive, the pollen or nectar quality will be evaluated. If the pollen or nectar quality is above threshold, the nectar or pollen will be deposited and the scout bee will perform a dance known as waggle dance. The dance serves as a communication medium to tell other bees the location of the flower patch (direction, distance, quality and quantity). From the dance, fitness of each patches can be evaluate relatively in term of quality and effort needed to harvest the food. After the dance, the scouts will employ other bees and go back to the flower patch to collect the nectar. More bees are employed to more promising patches (i.e. better quality, quantity and nearer to the hive) and less bees are employed to the less promising patches. The food level of each patch will be monitored continuously during the harvest. The information will be used in the next waggle dance. Inspired from the food foraging behavior of honey bees described above, Figure 2 shows the pseudo code of BA in its basic form.

During initialization, there are several parameters that need to be set prior to execution (*n, m, e, nep, nsp and ngh*). From a hive, there will be *n* number of scout bee, which will randomly choose a flower patch around the hive. From *n*, *m* best patches will be selected and from *m*, *e* elite patches will be sent with *nep* bees while *nsp* bees sent to the rest of the patches (*m-e*). *nsp* and *nep* are referred as employed bees. Lastly, the size of neighborhood search, *ngh*, is to be determined.

1. Initialize population with random solutions
2. Evaluate fitness of the population.
3. While (stopping criterion not met)
   //Forming new population.
4. Select sites for neighborhood search.
5. Recruit bees for selected sites (more bees for best e sites) and evaluate fitnesses.
6. Select the fittest bee from each patch.
7. Assign remaining bees to search randomly and evaluate their fitness's.
8. End While.

Figure. 1. Pseudo code of basic Bees Algorithm

The algorithm starts with sending out scout bees randomly to the food source (i.e. search space). In step 2, each evaluation of patches visited by the scout bees are initialized for the first time. Step 4 chooses m number of bees. The patches visited by the m bees will be used for neighborhood search later. In step 5, a number of bees (*nep*) are employed for neighborhood search for the each elite patches (*e*). For the rest of the patches (*m-e*), a number of bees (*nsp*) are employed for neighborhood search for each of the non-elite patches. Here, more bees are sent to more promising patches to ensure more detailed search done in the neighborhood compared to less promising patches. Bees are sent to non-elite patches to reduce the possibilities of local maxima among the elite patches. In step 6, bee with the highest fitness will be selected, which will reduce the search space size. Finally in step 7, the remaining bees are randomly assigned to new potential solution. Step 4-7 are repeated until stopping condition is met. Figure 3 summarizes all the parameters need to be initialized prior to execution.

Although BA is considered as a new swarm-based algorithm, BA has been used to solve many optimization problems. BA shows promising results in term of effectiveness, problem scale and performance published as in [24–28]. Although many of the BA implementation mentioned is functional optimization, BA is claimed suitable to solve not only functional, but also combinatorial optimization problems [24]. These factors motivate us to adopt BA for t-way test data generation strategy.

B. BA Strategy for generating SCA

Our BA strategy for generating test data is divided into 2 main parts. The first part generates the sequence interaction set for the intended strength. The sequence interaction generation algorithm is designed to be generic, to support any strength, t, for N number of events, given $t < N$. This part consists of a simple function that generates permutation set of the intended events. If constraints are defined, sequences with invalid sequence will be marked as already covered. The second part runs the BA to generate candidate test cases. From the candidate test cases, the best test cases will be added to the final test suite. Should there be any constraints defined, the constraint will be considered during test case generation.

VI. EXPERIMENTS AND RESULTS

We implement our BA Strategy using JAVA Netbeans 7.0 IDE on Windows XP SP3 platform running on Intel Core2 duo 3.00 GHz with 2GB RAM. We studied several published implementation of BA in several problems [25–29] and set the parameters of our BA Strategy as in Table 2. We compare the SCA produces by our algorithm against t-seq, ASP and BA. The results are as in Table 3.

TABLE II. BA STRATEGY PARAMATERS

Parameter	Value	Description
n	20	Number of random test case generated
m	5	Selected test case with best coverage for neighborhood search
e	2	Elite test cases
nep	15	Number of improvement attempt for each elite test case
nsp	5	Number of improvement attempt for each non-elite site
ngh	4	Repetition for nep and nsp improvement

TABLE III SCA GEERATED FOR 3-WAY AND 4-WAY SEQUENCE

No. of event	3-way sequence		
	t-seq	ASP	BA
5	8	**7**	**8**
6	10	**8**	**8**
7	12	**8**	10
8	12	**8**	10
9	14	**9**	12
10	14	**10**	12
11	14	**10**	13
12	16	**10**	15
13	16	**10**	14
14	16	**10**	16
15	18	**10**	16
16	18	**11**	17
17	20	**11**	18
18	20	**-**	18
19	22	**-**	18
20	22	**19**	**19**
21	22	-	**20**
22	22	-	**21**
30	26	23	**23**
40	32	27	**26**
50	34	31	**30**

No. of event	4-way sequence		
	t-seq	ASP	BA
5	29		26
6	36		35
7	46		41
8	50		50
9	58		57
10	66		64
11	70		69
12	78		77
13	86		81
14	90		86
15	96		91
16	100	-	97
17	108	-	99
18	112	-	104
19	114	-	107
20	120	**104**	**104**
21	126	-	**116**
22	128	-	**120**
30	156	149	**145**
40	182	181	**175**
50	214	-	**190**

From the results, our BA strategy produces comparable results against *t-seq* and ASP. For small number of events, ASP produces the smallest SCA size, while BA performs better when the number of event reached 20 and above.

VII. CASE STUDY: SCA TESTING WITH CONSTRAINT SUPPORT

In order to demonstrate the applicability of BA strategy in generating SCA, we used the same real world problem described by Kuhn [14]. The SUT described as a laptop with 5 different devices (indicated as P1 to P5) that need to be connected to it. The system need to be booted up, run an application and perform device scan function. The behavior of the system is influenced by the device connection sequence. There are several constraints that need to be considered in testing the system. The system must be booted up first. Application must be run before device scan can be performed. Peripherals P1 to P5 can be connected in any sequence. We generated test cases with all the constraint are taken into consideration. We run our BA strategy to produce the test case for the mentioned scenario above. Since the system needs to be booted up before any other event occurs, the first event (system boot) is omitted and added to the test cases later. From the test suite generated using BA Strategy as in Table 4, it is shown that the strategy able to produce test suite size which is comparable to ASP (test suite size - 8) and tseq (test suite size -18). The test suite size is slightly higher than the pre-computed test suite since all of the constraints have been addressed during the test suite generation. In the current version of our strategy, we do not support the events with customize attribute as discussed in [14] and [15]. This is due to our strategy is designed to be a generic and not customize to fixed one specific scenario.

VIII. CONCLUSION AND FUTURE WORKS

In this paper we discussed the generation of test suite for event driven system. We reviewed existing approach and proposed a new approach called BA strategy based on Bees Algorithm. We conducted experiments and case study and compare with the existing approach. From the experiment and case study, although not the best, our BA Strategy performed well in some scenario compared to the other. We are looking forward to improve the strategy to further support more features such as variable strength and non-uniform value of events.

Table 1. Final Test Suite for 8 Steps Procedure

TC	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8
1	Boot	App	P1	P2	P5	P3	P4	Scan
2	Boot	P2	App	Scan	P4	P3	P1	P5
3	Boot	P4	P3	P5	P1	App	Scan	P2
4	Boot	P1	P5	P3	P4	P2	App	Scan
5	Boot	P4	App	P5	Scan	P3	P2	P1
6	Boot	P5	P2	P1	App	Scan	P4	P3
7	Boot	P2	P3	App	P5	P4	Scan	P1
8	Boot	App	Scan	P1	P4	P2	P5	P3
9	Boot	P2	P1	App	P3	Scan	P5	P4
10	Boot	P3	P2	P5	P4	App	P1	Scan
11	Boot	P4	App	Scan	P1	P5	P3	P2
12	Boot	App	P3	Scan	P2	P4	P5	P1
13	Boot	P5	App	Scan	P3	P1	P2	P4

ACKNOWLEDGMENTS

This research is partially funded by myGrants: A New Design of An Artifact-Attribute Social Research Networking Eco-System for Malaysian Greater Research Network, UMP RDU Short Term Grant: Development of a Pairwise Interaction Testing Strategy with Check-Pointing Recovery Support, and ERGS Grant: A Computational Strategy for Sequence Based T-Way Testing.

REFERENCES

[1] Bob Jenkin, Jenny [Online]. Available: http://burtleburtle.net/bob/math/jenny.html. [Accessed: 21-Jun-2013].

[2] 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, 2007. ECBS '07. 14th Annual IEEE International Conference and Workshops on the*, 2007, pp. 549 –556.

[3] Y. M. I and Z. K. Z, *MC-MIPOG: A Parallel t-Way Test Generation Strategy for Multicore Systems*, vol. 32. Taejon, COREE, REPUBLIQUE DE: Electronics and Telecommunications Research Institute, 2010.

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

[5] A. W. Williams, "Determination of Test Configurations for Pair-Wise Interaction Coverage," in *Proceedings of the IFIP TC6/WG6.1 13th International Conference on Testing Communicating Systems: Tools and Techniques*, Deventer, The Netherlands, The Netherlands, 2000, pp. 59–74.

[6] X. Chen, Q. Gu, A. Li, and D. Chen, "Variable Strength Interaction Testing with an Ant Colony System Approach," in *Software Engineering Conference, 2009. APSEC '09. Asia-Pacific*, 2009, pp. 160 –167.

[7] J. Czerwonka, "Pairwise Testing In real World," in Proceedings of 24th Pacific Northwest Software Quality Conference, 2006.

[8] B. S. Ahmed, K. Z. Zamli, and C. P. Lim, "Constructing a T-Way Interaction Test Suite Using the Particle Swarm Optimization Approach," *International Journal of Innovative Computing, Information and Control (IJICIC)*, vol. 8, no. 1, pp. 1–10, Nov. 2011.

[9] 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," *Inf. Softw. Technol.*, vol. 54, no. 6, pp. 553–568, Jun. 2012.

[10] W. Ziyuan, N. Changhai, and X. Baowen, "Generating combinatorial test suite for interaction relationship," in *Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting*, New York, NY, USA, 2007, pp. 55–61.

[11] Z. Wang, 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, pp. 155 –160.

[12] J. Arshem, "Test Vector Generator." [Online]. Available: http://tvg.sourceforge.net/.

[13] Rozmie R. Othman and Kamal Z. Zamli, "ITTDG: Integrated T-way test data generation strategy for interaction testing," *Scientific Research and Essays*, vol. 6, no. 17, pp. 3638–3648, Aug. 2011.

[14] D. R. Kuhn, J. M. Higdon, J. F. Lawrence, R. N. Kacker, and Y. Lei, "Combinatorial Methods for Event Sequence Testing," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, 2012, pp. 601 –609.

[15] Erdem Esra, Katsumi Inoue, Johannes Oetsch, Jorg Puhrer, Hans Tompits, and Cemal Yilmaz, "Answer-set programming as a new approach to event-sequence testing," presented at the The Third International Conference on Advances in System Testing and Validation Lifecycle (VALID 2011), Barcelona, Spain, pp. 26–34.

[16] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing," in *Proceedings of the 28th Annual International Computer Software and Applications Conference - Volume 01*, Washington, DC, USA, 2004, pp. 72–77.

[17] D. M. Cohen, S. R. Dalal, A. Kajla, and G. C. Patton, "The Automatic Efficient Test Generator (AETG) system," in *Software Reliability Engineering, 1994. Proceedings., 5th International Symposium on*, 1994, pp. 303 –309.

[18] R. Mandl, "Orthogonal Latin Squares: An Application of Experiment Design to Compiler Testing," *Commun. ACM*, vol. 28, no. 10, pp. 1054–1058, 1985.

[19] R. Brownlie, J. Prowse, and M.S. Phadke, "Robust Testing of AT&T PMX/StarMAIL using OATS," *AT&T Technical Journal*, vol. 71, no. 3, pp. 41–47, 1992.

[20] C. J. Colbourn and M. B. Cohen, "A Deterministic Density Algorithm for Pairwise Interaction Coverage," in *Proc. of the IASTED Intl. Conference on Software Engineering*, 2004, pp. 242–252.

[21] Alan Webber Williams, "Coverage Measurement And Generation of Configurations," Universiti of Ottawa, Ottawa ON Canada, 2002.

[22] G. Seroussi and N. H. Bshouty, "Vector sets for exhaustive testing of logic circuits," *Information Theory, IEEE Transactions on*, vol. 34, no. 3, pp. 513 –522, May 1988.

[23] C. Cheng, A. Dumitrescu, and P. Schroeder, "Generating Small Combinatorial Test Suites to Cover Input-Output Relationships," in *Proceedings of the Third International Conference on Quality Software*, Washington, DC, USA, 2003, p. 76–.

[24] D.T. Pham, A. Ghanbarzadeh, E.Koc, S.Otri, S.Rahim, and M.Zaidi, "The Bees Algorithm - A Novel Tool for Complex Optimization Problems," in *Proceedings of IPROMS 2006 Conference*, 2006.

[25] S. Anantasate and P. Bhasaputra, "A multi-objective bees algorithm for multi-objective optimal power flow problem," in Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2011 8th International Conference on, 2011, pp. 852 – 856.

[26] D. T. Pham, M. Castellani, and A. A. Fahmy, "Learning the inverse kinematics of a robot manipulator using the Bees Algorithm," in *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, 2008, pp. 493 –498.

[27] D. T. Pham, S. Otri, A. Ghanbarzadeh, and E. Koc, "Application of the Bees Algorithm to the Training of Learning Vector Quantisation Networks for Control Chart Pattern Recognition," in *Information and Communication Technologies, 2006. ICTTA '06. 2nd*, 2006, vol. 1, pp. 1624 –1629.

[28] D. T. Pham and M. Kalyoncu, "Optimisation of a fuzzy logic controller for a flexible single-link robot arm using the Bees Algorithm," in *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*, 2009, pp. 475 –480.