SPLBA: An Interaction Strategy for Testing Software Product Lines Using the Bat-Inspired Algorithm

CONFERENCE PAPER · AUGUST 2015

DOI: 10.13140/RG.2.1.2889.4169

Yazan Alsariera

Universiti Malaysia Pahang

7 PUBLICATIONS 1 CITATION

SEE PROFILE

Kamal Z Zamli

Universiti Malaysia Pahang

120 PUBLICATIONS 396 CITATIONS

SEE PROFILE

SPLBA: An Interaction Strategy for Testing Software Product Lines Using the Bat-Inspired Algorithm

Yazan A. Alsariera, Mazlina A. Majid and Kamal Z. Zamli
Faculty of Computer Systems and Software Engineering,
Universiti Malaysia Pahang
26300 Kuantan, Pahang, Malaysia
kamalz@ump.edu.my

***Abstract*—Software product lines (SPLs) represent an engineering method for creating a portfolio of similar software systems for a shared set of software product assets. Owing to the significant growth of SPLs, there is a need for systematic approach for ensuring the quality of the resulting product derivatives. Combinatorial t-way testing (where t indicates the interaction strength) has been known to be effective especially when the number of product's features and constraints in the SPLs of interest are huge. In line with the recent emergence of Search based Software Engineering (SBSE), this article presents a novel strategy for SPLs tests reduction using Bat-inspired algorithm (BA), called SPLBA. Our experience with SPLBA has been promising as the strategy performed well against existing strategies in the literature.**

***Keywords*—software testing; bat algorithm; constrained testing; nature inspired meta-heuristic algorithms**

I. Introduction

Software product lines (SPLs) relates to a set of software system that shares a set of common features [1]. Typically, SPLs is represented using a feature model, which represents the system configurations [2]. SPLs relationships are categorized into four main types; *compulsory*, *or*, *optional* and *alternative* [3] (see Fig. 3). To model dependencies, an excludes and requires relationships are introduced as part of the feature model [4].

SPLs testing is painstakingly difficult [5] owing to large combination of features and their dependencies [3, 6, 7]. Considering the smart mobile phone example in Fig. 1, even without considering constraints and dependencies, there are already 18 features to be considered. As each of these features can takes 2 values, there are already 2^{18} possible combinations to be considered for testing. Here, t-way testing (or termed t-wise testing), where t indicates the interaction strength, can be adopted to minimize the test data for consideration.

The work on t-way testing and its corresponding test generation strategies are now new [8]. Many existing works do exist taking up different approaches from algebraic to computational [9]. In line with the emergence of the field called Search based Software Engineering (SBSE), many researchers have started to opt for meta-heuristic algorithm including that of Particle Swarm Optimization (PSO) [10], Ant Colony Algorithm (ACA) [11], Genetic Algorithm (GA) [12, 13], Simulated Annealing (SA) [14] and Late Acceptance Hill Climbing (LAHC) [15] as the base algorithm for t-way strategy. Many reported results in the literature suggest that meta-heuristic based algorithms tend to be more superior as compared to other existing approaches. For this reason, we have opted to explore the use of meta-heuristic algorithms further. Specifically, owing to its promising performance [16, 17], we have decided investigated to adopt the Bat-inspired Algorithm (BA) for our work.

Going back to the example of the smart mobile system in Fig. 1, the main system components consist of calls, screen, messages, media and GPS.

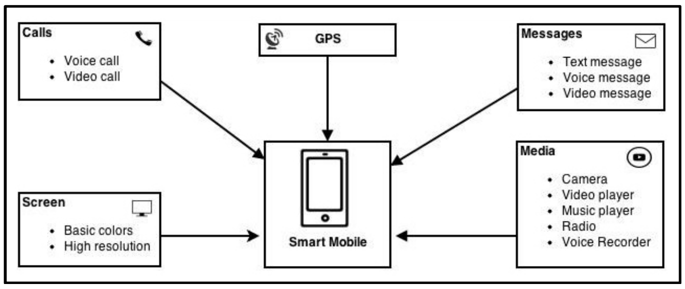

Fig. 1. Software product line of a smart mobile system.

The smart mobile system offers wide factors and levels (i.e., also termed parameters and values) depending on the features provided by the mobile device. Generally, software product lines are represented using feature model diagram. A feature model diagram is a tree structure to capture the relationship among different features. There are four types of relationship namely *optional*, *compulsory*, *alternative*, *or* as well as two composition rules called *requires* and *excludes* respectively. Furthermore, a feature may include cross-tree constraints that are explicitly expressed by the user.

Referring to the tree structure for the smart mobile system in Fig. 2 as an example, the semantic for *optional* implies that the given feature is optional whilst the semantic of *compulsory* dictates the necessary presence of the given feature. Meanwhile, the semantic of *optional* is such that at least one or all combinations of the given features can be selected. As for the semantic of *alternative* (i.e. *XOR*), only one feature must be selected from a combination of features. Finally, *requires* dictates the need of a particular feature to co-exist with the given feature of interest and *excludes* prescribes elimination of the combination of the given features.

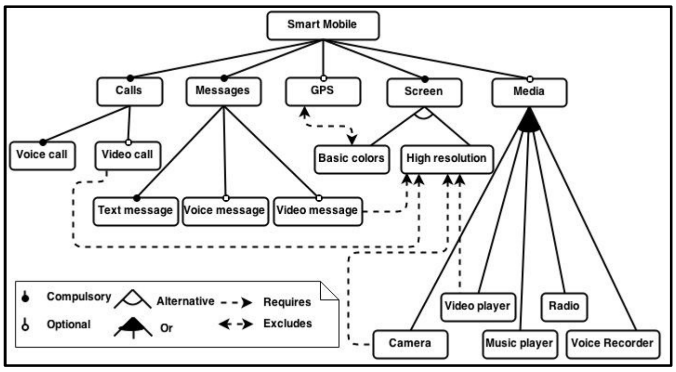

Fig. 2. The illustrative of the feature model for the smart mobile system.

Concerning the smart mobile systems, there are 13 system features of interest including voice call, video call, GPS, basic color screen, high resolution screen, test message, voice message, video message, camera, video player, music player, radio and voice recorder. Some features are deemed as constraints, for example, basic color screen and high resolution screen must not co-exist together. Similarly, four other features; video call, video message, camera and video player must be implemented with high screen resolution. Voice call and text messages are compulsory features whilst other features are optional.

System configurations	Values	
All 1-value parameter (smart mobile system, calls, messages, GPS, screen, media, voice call, test message)	True	
Video call	True	False
Voice message	True	False
Video message	True	False
Basic colors	True	False
High resolution	True	False
Camera	True	False
Video player	True	False
Music player	True	False
Radio	True	False
Voice recorder	True	False

Fig. 3. The smart mobile system configurations.

The complete analysis of the smart mobile system suggests two groups of features as seen in Fig.3. Specifically, there are one group of features with 1-valued parameter (smart mobile system, calls, messages, GPS, screen, media voice call and text message), and the other group with ten 2-valued parameter (video call, voice message, video message, basic color screen, high resolution screen, camera, video player, music player, radio and voice recorder).

Exhaustively, there are ($2^{10} \times 1^{8} = 1024$) possible configurations that can be generated for the smart mobile system. Here, the configuration grows with increasing number of features. Thus, the main questions are:

- How to minimize the tests?
- How to test the interaction between features?

In the literature, researchers are addressing the two aforementioned questions through the adoption of t-way testing (where t represents the interaction strength). Using a t-way testing implementation (or strategy), the total number of tests can be systematically reduced, that is, by relaxing the value of *t* interaction.

Much useful progress has been achieved so far; however, there are still some rooms for improvements. Firstly, much existing work has not given much focus on the support for constraints interaction with high number of parameters. Secondly, existing work also has not sufficiently considered the adoption of Bat Algorithm as the basis of the t-way implementation in line with the current hype on the new field called Search based Software Engineering.

Addressing the aforementioned issues, we have developed a new t-way strategy for SPLs based on the Bat-inspired algorithm, called SPLBA. Our research benchmarking experiment with ($t = 2$) against existing strategies using feature models from the SPLOT [18, 19] repository has been encouraging. SPLBA serves as our research vehicle to investigate the usefulness of adopting Bat Algorithms for highly constraints and high parameters SPLs.

The paper is organized as follows. Section II discusses covering array notation along with its usage for constraints SPLs. Section III discusses related work. Section IV presents the implementation of SPLBA testing strategy for features model. Section V reports experimental results. Section VI concludes this paper and discusses future work.

II. Covering Arrays and Their Applications

Mathematically, covering arrays (CA) has four parameters; N, t, p, and v (i.e. CA (N, t, v^p). Here, the symbols p, v, and t are used to refer to number of parameters, values, and interaction strength for the CA, respectively. For example, CA (9, 2, 3^4) represents a test suite consisting of 9×4 arrays (i.e., the rows represent the size of test cases (N), and the column represents the parameter (p)). In this case, the test suite also covers two-way interaction for a system with four 3-value parameters. As a special case, the covering array can be rewritten as CAN (N,t, v^p) when the CA carries the most optimal result.

Similar to CA, mixed covering array (MCA) has three parameters; N, t, and Configuration (C) (i.e. MCA (N, t, C)). Apart from N and t that carry the same meaning as in CA, MCA adopts a new symbol, C. In this case, C represents the parameters and values of each configuration in the following format: $v_1^{p1}\ v_2^{p2} \ldots\ v_n^{pn}$ indicating that there are p1 parameters with v_1 values, p_2 parameters with v_2 values, and so on. For example, MCA (1265, 4, $10^2\ 4^1\ 3^2\ 2^7$) indicates the test size of 1265 that covers four-way interaction. Here, the configuration takes 12 parameters (two 10-value parameters, one 4-value parameter, two 3-value parameters, and seven 2-value parameters).

Using the mixed covering array notation, the smart mobile system configuration can be represented as MCA (N; 2, 1^8 2^{10}). Considering t = 2, there are 45 possible 2-way interaction groups between all the piers in the system, where each pair has four configurations. So the total configurations are (45 × 4 = 180). However, there are some constraints on the feature model which should not be observed by the tests. These constraints include [(x, high resolution screen = true, basic color screen = true), (x, high resolution screen = false, basic colors screen = false), (x, voice call= false, test message = false), (x, voice call= true, test message = false), (x, voice call= false, test message = true) ,(x, basic color screen = true, video call = true, video message = true, camera= true, video player = true)]. A simple demonstration is described in the Fig. 4 for t=2 where the title field *1 represents all 1-value parameters (smart mobile system, calls, messages, GPS, screen, media voice call and text message).

		t = 2									
#	*1	video call	voice message	video message	basic colors	high resolution	camera	video player	music player	radio	voice recorder
1	True	False	True	False	True	False	False	False	False	False	False
2	True	False	False	False	True	False	False	False	False	False	False
3	True	False	True	False	True	False	False	False	False	True	False
4	True	False	True	False	True	False	False	False	False	False	True
5	True	False	True	False	False	True	True	False	False	False	False
6	True	False	True	False	False	True	False	True	False	False	False
7	True	False	True	False	False	True	True	True	False	False	False
8	True	True	True	False	False	True	True	True	False	False	False
…	.	.	.	.	.	.	.	.	.	.	.
180	True	True	True	True	False	True	True	True	True	True	True

Fig. 4. The illustrative of the smart mobile system interactions.

The constraints (or unwanted combinations) highlighted earlier must be removed from the interaction tuples. The system configuration can be formally expressed using the mixed-constraints covering array notation discussed on [20] as MCCA (MCA (N; 2, 1^8 2^{10}), F), where F= [(True, True, True, True, True, True, True, False, x, False, True, False, False, False, x, x, x), (True, True, True, True, True, True, True, x, x, x, False, True, x, x, x, x, x)]. (i.e., condition = [(calls, messages, GPS, screen, media, voice call, test message, video call, voice message, video message, basic colors, high resolution, camera, video player, music player, radio, voice recorder)]).

The complete test suite for the 2-way interaction is illustrated in Fig. 5. Here, the constraints pairs are eliminated from the test combinations (i.e., basic colors and high resolution screen can't be included in the same test set. Also, video call, video message, camera and video player are not allowed to be true value (or included) when a basic color screen is true. The final result is 41 pairs. So the total configurations after removing the constraints are (41 × 4 = 164).

	Video call	Voice message	Video message	Basic colors	High resolution	Camera	Video player	Music player	Radio	Voice recorder
Video call	0	1	1	0	1	1	1	1	1	1
Voice message	0	0	1	1	1	1	1	1	1	1
Video message	0	0	0	0	1	1	1	1	1	1
Basic colors	0	0	0	0	1	0	0	1	1	1
High resolution	0	0	0	0	0	1	1	1	1	1
Camera	0	0	0	0	0	0	1	1	1	1
Video player	0	0	0	0	0	0	0	1	1	1
Music player	0	0	0	0	0	0	0	0	1	1
Radio	0	0	0	0	0	0	0	0	0	1
Voice recorder	0	0	0	0	0	0	0	0	0	0

Fig. 5. Pairs for consideration with constraints for t=2

Unlike conventional t-way testing, SPLs have two unique characteristics; test parameters are Boolean parameters, and a large number of constraints usually occur in the feature model. In such a case, this is one common approach for t-way testing to model the configurations as Boolean parameters ones (or zeroes) indicate the status of include (or exclude) [3], and the features as true (or false).

III. Related Work

In this section, we discuss related work on testing of SPLs, combinatorial testing, and constraints t-way testing. Existing interaction strategies for SPLs testing can be classified into two classifications; computational approaches which use an automated generation algorithm to construct the test suite. Unlike, Algebraic approaches are constructing test sets using mathematical properties of covering arrays [3].

Recently, several strategies and approaches have proposed using combinatorial testing for SPLs. Machado et al [21] has reviewed the strategies for SPLs combinatorial testing from 1998 to 2012. The survey shows that the main challenge for the combinatorial testing strategies was handling constraints. The use of constraints solver such as ICPL [22, 23] has slightly improved the handling of constraints. Perrouin et al. [24] proposed a t-way test generation strategy using three algorithms; solveCT, binSplit and incGrow. solveCT is an SAT solver based uses MiniSAT or ZChaff to solve constraints generations in the interaction tuples. binSplit algorithm is in charge of splitting the valid tuples to a solvable subset. Finally, incGrow algorithm builds the final tests incrementally. SPLCAT [25] adopts a covering array generation where the rows represent the product configurations while the columns be the model features. SPLCAT is covering the feature combinations by adding configurations incrementally, where the maximum number of interactions is tried to be covered by each added configuration.

Pairwise Independent Combinatorial Testing (PICT) [26] is the public domain strategy implementation developed by Microsoft, which addresses constraints. Adopting random selection for completing the uncovered pair interaction, PICT often offers non-optimal results. Unlike, LOOKUP [3] considered a superior t-way test generation tool with integrated constraints handling strategy, it uses IPOG algorithm as a backbone engine in combining with minimum invalid tuples (MIT) to checking validity. It selects the first parameters for (t) interaction, then generated the t-way tests, then extends for more parameters, and continues the process until all the t-way tests are covered, which adopts a general IPOG-C [27] test generation algorithm. IPOG-C uses constraint solvers like ICPL [22, 23]. However, PICT and LOOKUP use forbidden tuples as constraints handler strategy. All the necessary forbidden tuples are generated using constraints input. Then match them with the generated test cases.

SA_SAT [28] a variant one of Simulated Annealing via the metal re-heat and cooling metaphor method, SA_SAT depends on a large random search space for generating a pairwise test suite. Using probability-based transformation equations, SA_SAT implements a binary search algorithm to find the best test case per iteration of which is being added to the final test suite. Here, the selection of the best test case per iteration also takes into account the presence of constraints (i.e. upon finding one. The iteration will generate new candidate). In the reported work, SA_SAT addresses the support for constraints. Similar to SA_SAT, mAETG_SAT [29] which enable constraint testing. Empirical evidence suggests that both successful use for SPLs testing [30]. mAETG_SAT adopts integrated SAT solver for handling constraints. However, mAETG_SAT is not capable of handling thousand-sized Future models (FMs). Where it didn't scale well to FMs of over 200 features, according to the experiments on [31].

IV. DESIGN AND IMPLEMENTATION OF SPLBA

In a nutshell, The Bat Algorithm (BA)[32] is natural-inspired meta-heuristic algorithm proposed by Yang is 2010. Natural-inspired algorithms are based on perception of the nature. As an interpretation of the nature, they typically are not perfect [33]. BA is a population optimization algorithm founded on the hunting behavior of Microbats by using echolocation. BA is combining swarm and a path algorithm. BA provides a swarm behavior where it takes into account several bats (populations) and their positions and velocities at predefine dimensions to find the best solution. This behavior is an instant of PSO algorithm, which using swarm's population to find its solution. Moreover, BA provides an exhaustive local search method throughout it is random walk behavior around the best solution founded in each iteration cycle. This behavior is inspired from path algorithms, which use a local search around the appropriate entities to find the best overall. Furthermore, BA adopts a similar path method known on simulated annealing (SA), which are some variables such as bats size and emission pulse rates intensely behave like the temperature and the cooling factor on SA.

Typically, BA has been built on the assumption that the bat is able to find its prey in complete darkness. Applying this meta-heuristic algorithm for t-way strategy, we can see the test cases as bats position which is a possible solution of the problem. Searching by the algorithm provides a best test case (or global optimum) which indicates the quality of the solution by the best bat position to it is pray. Bats are avoiding obstacles using echolocation. In such a case, different frequencies are returned.

```
[1].  Objective function $f(x_i)$, $_{xi} = (x_{i1}....x_{ij})^T$
[2].  Initialize the bat population $x_i$ and velocities $v_i$ for i = (1,2..no bats)
[3].  Define pulse frequency $Q_i$ within $[Q_{min}, Q_{max}]$
[4].  Initialize pulse rates $r_i$ and the loudness $A_i$
[5].  While $(t<T_{max})$
            Generate new solutions by adjusting frequency, and update
            velocities and locations [Eq. 1 to 3 motion equations]
                        $f_i = f_{min} + (f_{max} + f_{min})\beta$            (1)
                        $v_i^{t+1} = v_i^t + (x_i^t - x_o)f_i$             (2)
                        $x_i^{t+1} = x_i^t + v_i^{t+1}$                 (3)
        if $(rand(0,1) > r_i)$
            Select the best solution in the current population
            Generate a local solution around the best solution
        End if

            if $(rand(0,1) < A_i$ and $f(x_i) < f(x))$
            Accept the new solutions
            Increase $r_i$ and reduce $A_i$
          End if
          Rank the bats and find the current best
      End while
[6].  Process results and visualization
```

Fig. 6. The pseudo code of the general Bat Algorithm [32].

The complete step of BA is shown in the pseudo code in Fig. 6. Briefly, for every bat *i* a position x_i and velocity v_i in a *D*- dimensional search space that is randomly initialized. After the initialization for the first set of position x_i and velocity v_i the fitness is calculated for each solution. Here, loudness can change from a large (positive) A_0 to a minimum value A_{min}. Furthermore, Bats are automatically adjusting the emitted pulses and adjusting the rate of pulse emission random r[0, 1], of their echolocation frequency. Those solutions are being updated and improved by each iterative using the equations 1-3 shown in Fig. 6 until the best fitness for the problem is found.

SPLBA strategy has adopted BA algorithm behavior. We have introduced population lists, which represents the interaction elements (IE) tuples for SPLBA strategy. The input is a pre-generated list contains all the IE for the targeted test suite. We can explain the strategy using three main processes; generation process, constraints-handler process in two stages and BA inspired based searching process.

As far generating process, SPLBA initializes pairing criteria to find all the binary pair combinations, then applying the first stage of the constraints elimination process for the possible constraints combination pairs. Upon completion generate the interaction elements tuples with all the possible test suite interaction elements.

The searching process continues until the interaction list is empty, which involves initialization of the bat population and mapping it to random test cases based on the selected interaction. Furthermore, define the frequency and the initial emission pulse of rate. Then SPLBA iterates each individual bat in search for the best fitness values. The test case represented in each individual bat is updated in each iteration, based on the updated individual position, frequency and velocity (see motion equations 1-3 in Fig. 6).

At the maximum iteration, or maximum fitness the best bat (or test case) will cover the maximum number of uncovered test interaction elements. Then, the second stage of the constraint process matches the best selected test case with constraints. In case of a match, the new local solution will randomly be generated, then select new best cases. Finally, the covered interaction elements are removed from the interaction elements list. Upon completion, the result of the final test suite is displayed. The pseudo code of the Bat-inspired strategy for SPLs (SPLBA) can be seen in Fig. 7.

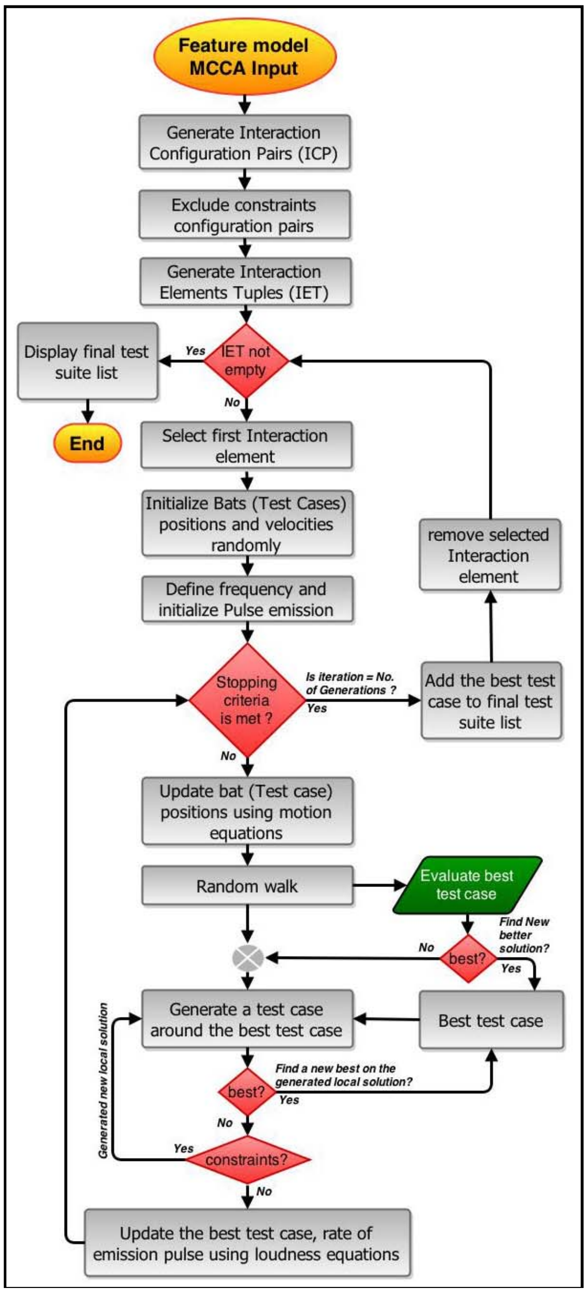
Feature model MCCA Input
Generate Interaction Configuration Pairs (ICP)
Exclude constraints configuration pairs
Generate Interaction Elements Tuples (IET)
Display final test suite list
Yes
IET not empty
No
End
Select first Interaction element
Initialize Bats (Test Cases) positions and velocities randomly
remove selected Interaction element
Define frequency and initialize Pulse emission
Stopping criteria is met ?
Is iteration = No. of Generations ?
Yes
Add the best test case to final test suite list
No
Update bat (Test case) positions using motion equations
Random walk
Evaluate best test case
Find New better solution?
No
best?
Yes
Generated new local solution
Generate a test case around the best test case
Best test case
Find a new best on the generated local solution?
best?
Yes
No
Yes
constraints?
No
Update the best test case, rate of emission pulse using loudness equations

Fig. 7. The illustrative of SPLBA flow chart.

V. Experimental Design and Results

Our experimental results goal is to benchmarking SPLBA against existing SPLs strategies to demonstrate its effectiveness. As far as verifying the accuracy and efficiency of our strategy is concerned, we run the optimization tests for 12 selected SPLs real systems provided by SPLOT [19] repository and compare the results provided by the literatures [3, 31]. We used HP workstation; Intel® Core ™ i7-3770 (3.40GHz, 3MB L3, 256KB L2, 32KB L1 cache), 4GB of RAM, Windows 7 professional Operating System with Java SE version 8 64bit.

The following experiment in (Table 1) takes the following attributes (or parameter) values. The population size number of bats = 200, number of generations = 200, loudness = 0.9, rate of pulse emission A = 0.9 in the frequency range of [0, 1] and tolerance = 0.025. We run SPLBA 20 times for each system configuration for statistical significance. The N/A field in the table represents not an available result in the selected configuration.

TABLE 1 Result of SPLBA 2-way interactions.

Feature Model	#Features	#Constraints	Strategies					
			ACTS (IPOG)	PICT	ICPL	SPLCAT	LOOKUP	SPLBA
Cellphone	11	22	9	N/A	N/A	8	N/A	**7**
Counter Strike Simple FM	24	35	13	N/A	N/A	10	N/A	**9**
SPL SimulES	32	54	11	N/A	N/A	10	N/A	**9**
DS Sample	41	201	103	N/A	N/A	97	N/A	**96**
Electronic Drum	52	119	35	N/A	N/A	27	N/A	**24**
Video Player	71	99	15	16	N/A	18	13	**10**
J2EE web arch	77	19	N/A	36	75	18	**17**	18
Billing	88	12	N/A	N/A	36	15	**13**	15
UP estructural	97	17	N/A	110	44	36	**34**	41
xtext	172	49	N/A	40	67	24	**17**	28
Eclipse1-Reuso	72	18	N/A	47	58	**19**	21	20
Coche ecologico	94	40	97	115	81	92	**90**	94

The experimental results demonstrate that SPLBA can compete with the existing strategies. Overall, SPLBA outperforms all the existing strategy in terms of test suite reduction for the first six systems; Cellphone, Counter Strike Simple FM, SPL SimulES, DS Sample, Electronic Drum and Video Player respectively. LOOKUP outperforms almost all strategies for the last 6 system configuration. SPLCAT excels for the case of Elipse1-Reuso.

Conclusion

In this research work, we have proposed a novel t-way strategy called SPLBA for SPLs testing, based on the Bat algorithm. Our experimental results are encouraging, especially at the prospect of supporting a high number of parameters. As the scope for future work, we are also working to support the high interaction strength (i.e, up to *t* =6) in order to capture more faults in SPLs system.

Acknowledgment

The work reported in this paper is funded by MOSTI eScience fund for the project titled: Constraints T-Way Testing Strategy with Modified Condition/Decision Coverage from the Ministry of Science, Technology, and Innovation, Malaysia. We thank MOSTI for the contribution and support.

References

[1] G. Böckle, F. J. van der Linden, and K. Pohl, *Software product line engineering: foundations, principles and techniques*: Springer Science & Business Media, 2005.

[2] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," DTIC Document1990.

[3] L. Yu, F. Duan, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Combinatorial test generation for software product lines using minimum invalid tuples," in *15th International Symposium onHigh-Assurance Systems Engineering (HASE)*, 2014, pp. 65-72.

[4] L. Bettini, F. Damiani, and I. Schaefer, "Implementing type-safe software product lines using parametric traits," *Science of Computer Programming,* vol. 97, Part 3, pp. 282-308, 1/1/2015 2015.

[5] S. Oster, I. Zorcic, F. Markert, and M. Lochau, "MoSo-PoLiTe: tool support for pairwise and model-based software product line testing," in *5th Workshop on Variability Modeling of Software-Intensive Systems*, 2011, pp. 79-82.

[6] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, Jr., "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering,* vol. 30, pp. 418-421, 2004.

[7] M. N. Borazjany, L. Yu, Y. Lei, R. Kacker, and R. Kuhn, "Combinatorial testing of ACTS: A case study," in *5th International Conference on Software Testing, Verification and Validation (ICST)*, 2012, pp. 591-600.

[8] D. R. Kuhn, D. R. Wallace, and J. AM Gallo, "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering* vol. 30, pp. 418-421, 2004.

[9] R. R. Othman, K. Z. Zamli, and S. M. Syed Mohamad, "T-Way testing strategies: A critical survey and analysis," *International Journal of Digital Content Technology and its Applications(JDCTA),* vol. Volume7, 2013.

[10] B. S. Ahmed and K. Z. Zamli, "A variable strength interaction test suites generation strategy using particle swarm optimization," *Journal of Systems and Software,* vol. 84, pp. 2171-2185, 2011.

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

[12] J. D. McCaffrey, "Generation of pairwise test sets using a genetic algorithm," in *33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09)*, 2009, pp. 626-631.

[13] J. D. McCaffrey, "An empirical study of pairwise test set generation using a genetic algorithm," in *7th International Conference on Information Technology: New Generations (ITNG)*, 2010, pp. 992-997.

[14] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn, "Constructing test suites for interaction testing," in *25th International Conference on Software Engineering*, 2003, pp. 38-48.

[15] A. A. Al-Sewari, K. Z. Zamli, and B. Al-Kazemi, "Generating t-way test suite in the presence of constraints," in *8th Malaysia University Conference Engineering Technology (MUCET)*, Melaka, Malaysia, 2014.

[16] K. Khan and A. Sahai, "A comparison of BA, GA, PSO, BP and LM for training feed forward neural networks in e-learning context," *International Journal of Intelligent Systems and Applications (IJISA),* vol. 4, p. 23, 2012.

[17] N. M. Sureja, "New inspirations in nature: a survey," *IJCAIT,* vol. 1, pp. 21-24, 2012.

[18] M. Mendonca, M. Branco, and D. Cowan, "SPLOT: software product lines online tools," in *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, 2009, pp. 761-762.

[19] M. Mendonca, M. Branco, and D. Cowan. (2009). *SPLOT: software product lines online tools*. Available: http://www.splot-research.org/

[20] M. B. Cohen, "Designing test suites for software interaction testing," Doctor of Philosophy Thesis, Computer Science, University of Auckland, University of Nebraska–Lincoln, 2004.

[21] I. do Carmo Machado, J. D. McGregor, and E. Santana de Almeida, "Strategies for testing products in software product lines," *ACM SIGSOFT Software Engineering Notes,* vol. 37, pp. 1-8, 2012.

[22] M. F. Johansen, Ø. Haugen, and F. Fleurey, "An algorithm for generating t-wise covering arrays from large feature models," in *16th International Software Product Line Conference*, 2012, pp. 46-55.

[23] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, "Using feature model knowledge to speed up the generation of covering arrays," in *7th International Workshop on Variability Modelling of Software-intensive Systems*, 2013, p. 16.

[24] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. Le Traon, "Automated and scalable t-wise test case generation strategies for software product lines," in *3rd International Conference on Software Testing, Verification and Validation (ICST)*, 2010, pp. 459-468.

[25] M. F. Johansen, Ø. Haugen, and F. Fleurey, "Properties of realistic feature models make combinatorial testing of product lines feasible," in *Model Driven Engineering Languages and Systems*, ed: Springer, 2011, pp. 638-652.

[26] J. Czerwonka, "Pairwise testing in the real world: Practical extensions to test-case scenarios," in *24th Pacific Northwest Software Quality Conference*, 2006, pp. 419-430.

[27] L. Yu, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Acts: A combinatorial test generation tool," in *6th International Conference on Software Testing, Verification and Validation (ICST)*, 2013, pp. 370-375.

[28] 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, pp. 129-139.

[29] M. B. Cohen, M. B. Dwyer, and S. Jiangfan, "Exploiting constraint solving history to construct interaction test suites," in *Testing: Academic and Industrial Conference Practice and Research Techniques (MUTATION, 2007)*, Taicpart, Mutation, 2007, pp. 121-132.

[30] M. B. Cohen, M. B. Dwyer, and J. Shi, "Interaction testing of highly-configurable systems in the presence of constraints," presented at the International Symposium on Software Testing and Analysis, London, United Kingdom, 2007.

[31] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. le Traon, "Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines," *IEEE Transactions on Software Engineering,* vol. 40, pp. 650-670, 2014.

[32] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," in *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, ed: Springer, 2010, pp. 65-74.

[33] X. Meng, X. Gao, and Y. Liu, "A novel hybrid bat algorithm with differential evolution strategy for constrained optimization," *International Journal of Hybrid Information Technology,* vol. 8, pp. 383-396, 2015.