International Journal of Computer and Information Technology (ISSN: 2279 – 0764)
Volume 9– Issue 4, July 2020

PWiseHA: Application of Harmony Search Algorithm for Test Suites Generation using Pairwise Techniques

Aminu Aminu Muazu[1]
Department of Computer Science
Umaru Musa Yar'adua University
Katsina, Nigeria.
Email: aminu.aminu [AT] umyu.edu.ng

Umar Danjuma Maiwada[2]
Department of Computer Science
Umaru Musa Yar'adua University
Katsina, Nigeria.
Email: umar.danjuma [AT] umyu.edu.ng

Abstract--- **Pairwise testing is an approach that tests every possible combinations of values of parameters. In this approach, number of all combinations are selected to ensure all possible pairs of parameter values are included in the final test suite. Generating test cases is the most active research area in pairwise testing, but the generation process of the efficient test suite with minimum size can be considered as one of optimization problem. In this research paper we articulate the problem of finding a pairwise final test suite as a search problem and the application of harmony search algorithm to solve it. Also, in this research paper, we developed a pairwise software testing tool called PWiseHA that will generate test cases using harmony search algorithm and this PWiseHA is well optimized. Finally, the result obtained from PWiseHA shows a competitive results if matched with the result of existing pairwise testing tools. PWiseHA is still in prototype form, an obvious starting point for future work.**

Keywords--- Software testing, Pairwise testing, interaction strength, Test suites, Harmony search algorithms.

I. INTRODUCTION

The entire aspiration of a software company is to ensure that a software is delivered with a high quality to it customers [1] [2]. Therefore, to achieve a high-quality software, the software need to be tested. The software testing makes sure that software achieves the user requirements, such that to avoid failures visible to customers. Testing is very important phases in software development lifecycle. Lack of testing may lead to harmful consequences which include the loss of an important data, the fortunes, and even the lives of people [3]. The main aim of software testing is to minimize the recognized software fault which is not accepted [4] [5] [6]. As such, the software engineers need only to consider a significant huge number of test data. Software testing refer to sequence of processes that was aimed to ensure that the software configuration does what it was planned to do and that it does not do something unplanned [7]. To test all likely combinations of inputs data and execution paths is called exhaustive testing, but it's beyond our reach [8].

Combinatorial testing refer to a specification based testing standard that requires for each *d*-way (where *d* indicates the combination strength degree) combination of input parameters of a given system, for every combination of a valid values of these *d* parameters can be covered by at least one test case [7] [9]. In pertinent literature, it is reported that *d*-way interaction small value of *d* is more effective in testing. Combinatorial testing is communal as a real technique to uncover accidental feature interactions which are confidential to a given software system. This tenacity has made clear that test cases are created by mingling tuples that are from different input parameters. This approach was recognized with success in terms of providing a very low cost testing in our true situations. The critical issues of testing a software is to detect software faults and generate an optimized test suite. Combinatorial testing provides probability for fault detection, which triggered by the interaction among parameters in the software under testing [10]. For this large interaction space, the exhaustive testing is generally impractical, even though there are available resources (tools) to do it, because most of the interaction values do not cause any failure. Combinatorial testing provides the smaller test suite that cover the large interaction parameter values [11] [12].

Combinatorial testing is an organized approach for sampling large provinces of test data. Observations have been found that most of the system faults are encountered when there is interactions between parameters values [7] [11]. This is the origin of pairwise definition, which can also refer to as 2-wise testing. In this techniques number of all combinations are selected set by set to ensure all possible pairs of parameter values are included in the final test suite. On the other hand, many evidences suggest that most software failures are caused by an unwanted pairwise interactions between the parameters of a system [13].

Only one issues is considered most in pairwise testing, that is, each pairwise interaction most be covered by at least one test in the final test suite [14].

Most of the time, testing all the test cases (exhaustively) is always impossible, this is because of timing constraints and also resources [15] [16]. The main problem is

In this research paper, we expressed the problem of generating pairwise test suites as a search problem and applying harmony search algorithm in solving the problem. Also, we introduced a pairwise tool called PWiseHA that may well serve as a configuration for generating pairwise test suites with harmony search algorithm.

We carry out a successions of experiments in order to evaluate the effectiveness and performance of our pairwise prototype tool normally known as PWiseHA. In lieu of the evaluation, we have used the available benchmark problems from the pairwise site [17]; it has presented list of many tools for generating pairwise test suits, where most of them included with their competence actions in terms of generating final test suites. Our tactic is more reasonable in that, if related with the existing pairwise tools in [17]. PWiseHA is still in a prototype form, which is an obvious starting point for future work would be to complete the implementation.

II. PROBLEM STATEMENT

Nowadays software systems can be run with different number of configurations, these configurations are made up of parameters and their respective values. All of these configurations need to be considered during testing.

The pairwise type of testing is a combinatorial testing skill that tests all possible pairs of input parameter values [18]. The most leading challenge in pairwise testing is that to find a test suites which consist of the smallest number of test cases that covers all pairs of input parameters of a software system [19]. Normally, in pairwise testing the capable way of finding a best solution is not predictable, because the time needed to generate the test cases grows promptly as the increased numbers of parameters with their respective values. Although, there are many existing pairwise strategies that minimized the number of test cases in a software system, but most of these strategies are not well optimized; rather they provide an acceptable solutions [20].

The exact research problem is therefore to develop a harmony search algorithm prototype tool which is proficient for creating and reducing as much as promising test suites that contain all pairs of input parameters values of the software under testing. The basics behind choosing the harmony search algorithm in PWiseHA is that it has the power to control the search between the local solutions and global solutions based on its parameters [20].

to reduce the quantity of combinations while keeping the effectiveness of detecting errors. Some number of techniques have been explored such that to address this particular problem. Therefore, pairwise is a better technique to prevent many of this problems.

III. RELATED WORK

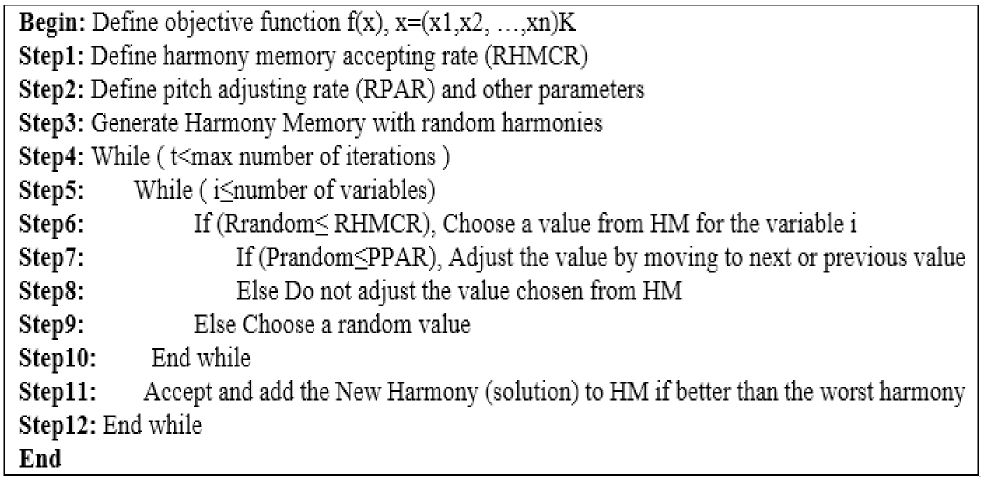
Software testing consumes most of the time and cost spent on software development. Basically, a lot of researchers developed different pairwise strategies to solve this problem and by generating an optimized test suite [4] [5].

The Automatic Efficient Test Generator (AETG) is a testing tool that constructs a test suite by adopting one test at a time approach using greedy algorithm [21]. The In-Parameter Order (IPO) is a testing tool strategy that constructs a test suite by adopting one parameter at a time using a horizontal and vertical algorithm [1] [12]. The Test Configuration for pairwise interaction (TConfig) is a testing tool that constructs a test suite using recursive algorithm [17]. The main idea for Jenny strategy is it start with one pair (itself) and it will then search for if there is other one pair, if there is not then it goes to two pairs. It will also search for if there are other 2-pairs, if not it will then go to 3-pairs. The same thing for other pairs. Therefore, the process will continue until all pairs are covered [17]. The Disability Discrimination Act (DDA) constructs a test suite using greedy algorithm [17]. The All-Pairs Testing (AllPairs) is a testing tool that constructs a test suite using greedy algorithm [17]. The Pairwise Independent Combinatorial Tool (PICT) constructs a test suite using core generation algorithm with random selection [17], PICT is not reliable to provide a non-optimal test size when compared with other strategies because of random conduct [14]. The rest: the Combinatorial Test Services (CTS) tool, the TestCover testing tool, the EXACT testing tool, the IPO family known as IPOS, the ecFeed testing tool, and the JCUnit testing tool uses some sort of deterministic algorithms [19].

IV. MATERIALS AND METHODOLOGY

A. *Harmony Search Algorithms for pairwise testing*

The Harmony search first begun when listening to an attractive piece of standard music. This is when musicians compose the harmony, they frequently try many possible combinations of the music pitches that are stored in the memory, which can be considered as an optimization process of adjusting the pitches (input) to obtain the optimal output (a perfect harmony). Harmony search draws the inspiration from harmony improvisation and has gained good result in the optimization area [4] [22].



```
Begin: Define objective function f(x), x=(x1,x2, …,xn)K
Step1: Define harmony memory accepting rate (RHMCR)
Step2: Define pitch adjusting rate (RPAR) and other parameters
Step3: Generate Harmony Memory with random harmonies
Step4: While ( t<max number of iterations )
Step5:     While ( i≤number of variables)
Step6:         If (Rrandom≤ RHMCR), Choose a value from HM for the variable i
Step7:             If (Prandom≤PPAR), Adjust the value by moving to next or previous value
Step8:             Else Do not adjust the value chosen from HM
Step9:         Else Choose a random value
Step10:    End while
Step11:    Accept and add the New Harmony (solution) to HM if better than the worst harmony
Step12: End while
End
```

Figure 1. Pseudocode of Harmony search algorithm [14].

The music improvisation is simply a way for searching a better harmony, and it's achieved when attempting different combinations of pitches that satisfies the following rules [23]**:**

- Playing any one pitch from the memory
- Playing an adjacent pitch of one pitch from the memory
- Playing a random pitch from the possible range

This method is represented in each variable selection of the Harmony Search algorithm. Similarly, it should follow any of the three rules below:

- Choosing any value from the Harmony Search memory
- Choosing an adjacent value from the Harmony Search memory
- Choosing a random value from the possible value range

The above mentioned three rules in the harmony search algorithm are maintain by the following parameters: Harmony Memory Considering Rate (HMCR) and Pitch Adjustment Rate (PAR) [22] [24].

The first step will initialize the harmony memory. Now the harmony memory consists of a number of randomly generated solutions to the optimization problem under consideration, whereby these test cases are generated using pairwise techniques.

The second step will improvise a new solution from the harmony memory. Each component of this solution is obtained based on the HMCR. Still here the test cases were generated using pairwise techniques.

The third step will update the harmony memory. Also new test pair which was generated from the second step is evaluated and if it's better than the worst in the harmony memory, it will replace it. Else, it will eliminated.

The fourth step will repeat second step to third step until a present termination criterion is met (i.e. the maximal number of iterations is met).

Harmony Search algorithm is in used successfully in an extensive variety of optimization problems. It present numerous advantages with respect to traditional optimization techniques [23]. Figure 1 displays the pseudocode of Harmony Search Algorithm.

B. *PWiseHA Implementation*

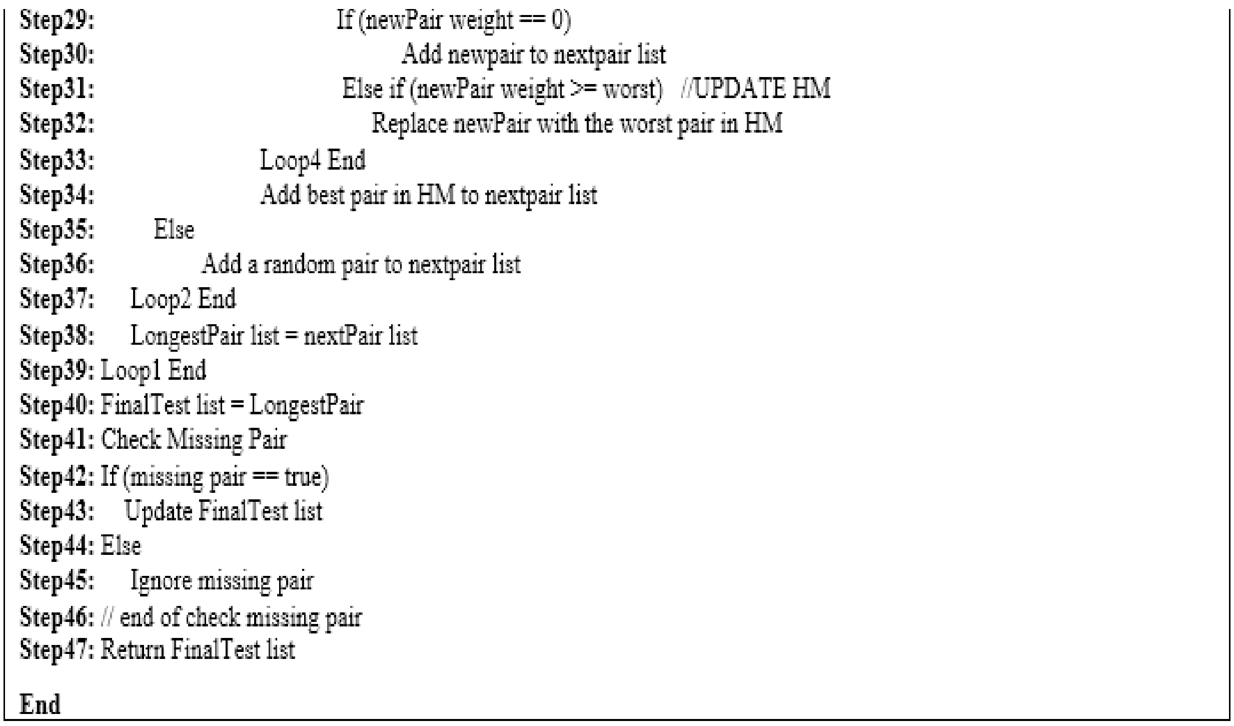
This prototype tool contain only one algorithm that will optimize and generate a near optimal final test suite. Here in this algorithm the concept and procedures in harmony search is applied that would work with pairwise testing techniques.
The steps used in PWiseHA are as follows:
*1st Step***:** Here it will initialize the harmony search memory.
*2nd Step***:** Here it will improvise a new solution from the harmony memory.
*3rd Step***:** Here it will update the harmony memory.
*4th Step***:** Here it will repeat second step to third step until a present termination criterion is met.

In Figure 2, we have displays the pseudocode of PWiseHA strategy. This PWiseHA prototype tool was developed using Java programming language (in JCreator LE 4.50 environment) and Java foundation classes AWT and Swing are used for the Graphical User Interface (GUI) with JDK 1.6. Here, the Java programming language has been chosen because it has the features of platform-independent (which means it has the ability to run the same program on different operating systems), java GUI is user friendly, and also it has rich API's for manipulation of array lists.

```
INPUT: Parameter values, Combination strength
OUTPUT: Optimized final test suites

Start

Step1: Declare LongestPair list= null, NextPair list= null, FinalTest list= null
Step2: Read file
Step3: Sort the Parameters in respect to their values
Step4: Select & generate parameter combination based on d
Step5: Generate LongestPair
Step6: Loop1//LOOP ALL PARAMETERS TO PICK EACH
Step7:     Loop2 //LOOP LONGESTPAIR TO PICK EACH PAIR
Step8:         If ( NextPair list is not empty)
Step9:         Select a pair from LongestPair and Put into HM //  five times
Step10:            Loop3
Step11:                Select each pair from the HM &  Measure the weight
Step12:            Loop3 end
Step13:            if (weight== 0)
Step14:                Add selected pair from HM to nextpair list
Step15:            Else // improvise. Ten (10) times
Step16:                Loop4
Step17:                    Initialize HMCR (ranges 0-1)
Step18:                    Declare newPair
Step19:                    If (HMCR<0.7)    //Do local improvisation
Step20:                      Initialize PAR (ranges 0-1)
Step21:                      If (PAR<0.9)  //adjust
Step22:                    newPair = Change the value of last parameter of  the selected test pair from
Step23:                      HM within the same parameter
Step24:                     else  //no adjust
Step25:                      newPair=selected pair from HM
Step26:                else        // Do global improvisation
Step27:                newPair=select random test pair from HM
Step28:                    Measure newPair weight
```

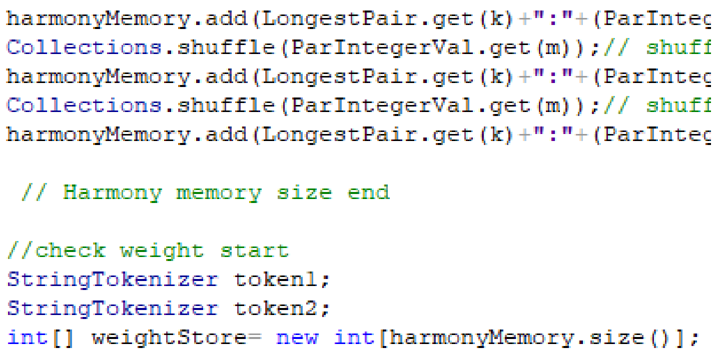


```
Step29:                     If (newPair weight == 0)
Step30:                         Add newpair to nextpair list
Step31:                     Else if (newPair weight >= worst)   //UPDATE HM
Step32:                         Replace newPair with the worst pair in HM
Step33:                 Loop4 End
Step34:                 Add best pair in HM to nextpair list
Step35:         Else
Step36:             Add a random pair to nextpair list
Step37:     Loop2 End
Step38:     LongestPair list = nextPair list
Step39: Loop1 End
Step40: FinalTest list = LongestPair
Step41: Check Missing Pair
Step42: If (missing pair == true)
Step43:     Update FinalTest list
Step44: Else
Step45:     Ignore missing pair
Step46: // end of check missing pair
Step47: Return FinalTest list

End
```

Figure 2. Pseudocode of PWiseHA

The PWiseHA prototype tool structure contains a set of GUI, a set of java classes, a simple database to store the final test suit result and Operating System (OS). All the components of PWiseHA strategy depends on the OS layer, which will run the overall prototype. Some fragment codes of PWiseHA were shown in Figure 3.



```
harmonyMemory.add(LongestPair.get(k)+":"+(ParInteg
Collections.shuffle(ParIntegerVal.get(m));// shuff
harmonyMemory.add(LongestPair.get(k)+":"+(ParInteg
Collections.shuffle(ParIntegerVal.get(m));// shuff
harmonyMemory.add(LongestPair.get(k)+":"+(ParInteg

 // Harmony memory size end

//check weight start
StringTokenizer token1;
StringTokenizer token2;
int[] weightStore= new int[harmonyMemory.size()];
```

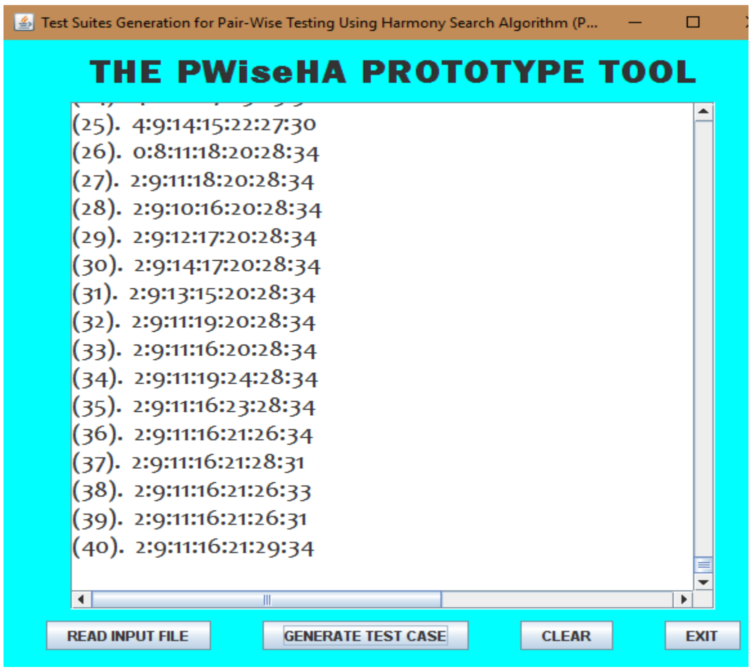
Figure 3. Some fragment codes of PWiseHA

The PWiseHA GUI's were developed in such away it has many attributes which requires a test engineer to read and execute the configuration test file and displays the final test suit. This attribute are illustrated in the TABLE I.

TABLE I. THE PWISEHA FORM ATTRIBUTES

Attribute Name	Type	Description
Read input file	Button	It will read a text file
Generate test case	Button	It will generate the final test suite
Clear	Button	It will clear all the text on text area
Exit	Button	It will close the window
	Text Area	It will display a result

The Figure 4 present the PWiseHA form with a displayed information of final test suite. Finally, the implementation of PWiseHA prototype tool has been demonstrated in fine points.


Test Suites Generation for Pair-Wise Testing Using Harmony Search Algorithm (P...
THE PWiseHA PROTOTYPE TOOL
READ INPUT FILE
GENERATE TEST CASE
CLEAR
EXIT

```
(25). 4:9:14:15:22:27:30
(26). 0:8:11:18:20:28:34
(27). 2:9:11:18:20:28:34
(28). 2:9:10:16:20:28:34
(29). 2:9:12:17:20:28:34
(30). 2:9:14:17:20:28:34
(31). 2:9:13:15:20:28:34
(32). 2:9:11:19:20:28:34
(33). 2:9:11:16:20:28:34
(34). 2:9:11:19:24:28:34
(35). 2:9:11:16:23:28:34
(36). 2:9:11:16:21:26:34
(37). 2:9:11:16:21:28:31
(38). 2:9:11:16:21:26:33
(39). 2:9:11:16:21:26:31
(40). 2:9:11:16:21:29:34
```

Figure 4. The PWiseHA form design with a displayed information of final test suite

V. RESULTS AND DISCUSSION

Evaluation of the PWiseHA mainly emphasized on the efficiency when generating a better final test suites sizes compared with existing strategies and the performance in terms of execution time when generating the final test suites.

We used the benchmark problems that are presented in pairwise testing website [17] to compare PWiseHA with those existing strategies. The website listed thirteen pairwise tools for generating final test suites, most of them with their efficiency measures [17]. These tools include: AETG, IPO, TConfig, CTS, Jenny, TestCover, DDA, AllPairs, PICT, EXACT, IPOS, ecFeed and JCUnit. The efficiencies of these tools were compared using six benchmark configurations with the notation x^y, where x represent the input parameters and y represent their distinct values. The configurations appears on different sizes, which are as follows: 3^4, 3^{13}, $4^{15}3^{17}2^{29}$, $4^{1}3^{39}2^{35}$, 2^{100}, and 10^{20}.

TABLE II: COMPARISON OF PWISEHA WITH SOME STRATEGIES IN [10] WHEN GENERATING BETTER AND BEST FINAL TEST SUITES

Configuration	**AETG**	**IPO**	**TConfig**	**CTS**	**Jenny**	**TestCover**	**DDA**	**AllPairs**	**PICT**	**EXACT**	**IPOS**	**ecFeed**	**JCUnit**	**PWiseHA**
3^4	9	9	9	9	11	9	?	9	9	9	9	10	10	**9**
3^{13}	15	17	15	15	18	15	18	17	18	15	17	19	23	**17**
$4^{15}3^{17}2^{29}$	41	34	40	39	38	29	35	34	37	?	32	37	49	**44**
$4^{1}3^{39}2^{35}$	28	26	30	29	28	21	27	26	27	21	23	28	33	**27**
2^{100}	10	15	14	10	16	15	15	14	15	10	10	16	18	**22**
10^{20}	180	212	231	210	193	181	201	197	210	?	220	203	245	**1048**

TABLE II displays the final test suite size of these tool, alongside the PWiseHA, it displays equal or sometimes better efficiencies on all of the benchmark configurations except that of the last two configurations.

TABLE III: EVALUATION PERFORMANCE OF PWISEHA IN TERMS OF EXECUTING TIME WHEN GENERATING FINAL TEST SUITES.

Configuration	PWiseHA **(Execution Time in sec)**
3^4	0.271
3^{13}	0. 326
$4^{15}3^{17}2^{29}$	18.250
$4^{1}3^{39}2^{35}$	9.232
2^{100}	3.810
10^{20}	45.600

Performance in terms of execution time is also measured (but no comparison), see TABLE III for the evaluation which is also good, because almost each of the configurations is executed in less than a minute which means it can save time during execution which will be able to lead to saving of cost as well.

VI. CONCLUSION

In this paper, we proposed and developed a strategy for pairwise testing called PWiseHA that generated an optimized final test suites by applying harmony search algorithm. Our evaluation results are inspiring, typically in terms of generating an optimal test suite in a suitable execution time. As part of our forthcoming work, PWiseHA is still in a prototype form, an obvious starting point for future work would be to complete the implementation.

ACKNOWLEDGMENT

This work is partly supported by a research grant of Tertiary Education Trust Fund (TETFund) from Umaru Musa Yar'adua University Katsina, Nigeria.

REFERENCES

[1] A. A. Muazu and A. A. Muazu. Design of a Harmony Search Algorithm Based on Covering Array T-Way Testing Strategy. *1st International Conference on Information Technology in Education & Development (ITED)*, ISBN: 978-978-35911-7-7, Page 33 – 38. April, 2018.
[2] X. Dianxiang, X. Weifeng, K. Michael, T. Lijo, and W. Linzhang. An Automated Test Generation Technique for Software Quality Assurance. *IEEE transactions on reliability*. VOL. 64, NO. 1. 2015.
[3] M. I. Younis, and K. Z. Zamli. A parallel t-way test generation strategy for multicore systems. *ETRI Journal*. 32(1), 73-83. 2010.
[4] A. B. Nasir, A. A. Alsewari, A A. Muazu and K. Z. Zamli. Comparative Performance Analysis of Flower Pollination Algorithm and Harmony Search based strategies: A Case Study of Applying Interaction Testing in the Real World. *2nd International Conference on New Directions in Multidisciplinary Research & Practice*, ISBN: 978-969-9948-47-3, 2016.
[5] A. A. Alsewari and K. Z. Zamli. Design and implementation of a harmony-search-based variable-strength t-way testing strategy with constraints support. *Journals on Information and Software Technology*. 54, 553–568. 2012.
[6] F. Konrad, and L. Horst. Combinatorial Robustness Testing with Negative Test Cases. *IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. DOI 10.1109/QRS.2019.00018. 2019.

[7] K. C. Ashis, C. Parna, C. Poulami and C. Aleena. Optimum Testing Time of Software using Size-Biased Concept. *ArXiv*: 1908.00307 v1 [*stat.ME*] 1 Aug 2019.

[8] B. Hambling. Software testing an ISTQB–ISEB foundation guide. 2011.

[9] K. Tai, and Y. Lie. Test generation strategy using pairwise. *IEEE transaction on software engineering*. 28(1), 109-111. 2002.

[10] S. G. Laleh, C. Jaganmohan, L. Yu, K. Raghu, K. Richard BEN: a combinatorial testing-based fault localization tool. *IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops* (ICSTW). 978-1-4799-1885-0. 2015.

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

[12] J. Yan, and J. Zhang. Combinatorial testing: principles and methods. *Journal of Software*. 20(6), 1393-1405. 2009.

[13] K. Z. Zamli, M. Klaib, M. Younis, N. Isa and R. Abdullah. Design and implementation of a t-way test data generation strategy with automated execution tool support information science. *Journal of information science*. 181(9), 1741-1758. 2012.

[14] A. A. Alsewari and K. Z. Zamli. A harmony search based pairwise sampling strategy for combinatorial testing. *International Journal of the Physical Sciences*. 7(7), 1062 - 1072. 2012.

[15] Y. Cui, L. Li and S. Yao. A new strategy for pairwise test case generation. *Third International Symposium on Intelligent Information Technology Application*. 3, 303-306. 2009.

[16] L. Z. Hasneeza, K. Z. Zamli. Migrating bird's optimization based strategies for pairwise testing. *9th Malaysian Software Engineering Conference*. 978-1-4673-82274. 2015.

[17] J. Czerwonka (2019, oct.) Pairwise testing, combinatorial test case generation. [Online]. Available: http://www.pairwise.org. 2019.

[18] A. Nahid and K. Susmita. Review Paper on Various Software Testing Techniques & Strategies. *Global Journal of Computer Science and Technology* Volume XIX Issue II Version I. 2019.

[19] F. Pedro and C. Yoonsik. "PWiseGen: Generating Test Cases for Pairwise Testing Using Genetic Algorithms". *IEEE International Conference on Computer Science and Automation Engineering (CSAE 2011), Shanghai, China*. 2011.

[20] A. A. Muazu and A. A. Muazu. One-Parameter-at-a-Time combinatorial testing Strategy Based on Harmony Search Algorithm Supporting Mixed Covering Array Mathematical Notation (OPATHS*). 1st International Conference on Information Technology in Education & Development (ITED),* ISBN: 978 978-35911-7-7, Page 33 – 38. April, 2018.

[21] D. M. Cohen, S. R. Dalal, A. Kajla and G. C. Patton. The automatic efficient test generator (AETG) system. *Journals on* International Symposium on Software Reliability Engineering *(IEEE* ISSRE*)*. 303-309. 1994.

[22] D. Manjarresa, I. Landa, S. Gil. A survey on applications of the Harmony search algorithm. *Eng. Appl. Artif. Intel.* 26(8), 1818–1831. 2013.

[23] Z. W. Geem, J. H. Kim, G. V. Li. A new heuristic optimization algorithm: Harmony search, Simulation 76 (2) (2001) 60–68. 2001.

[24] Z. W. Geem. (ed.), Music-Inspired Harmony Search Algorithm (*Springer, Berlin, 2001*). 2001.