One-Parameter-At-A-Time Combinatorial Testing Strategy Based on Harmony Search Algorithm Supporting Mixed Covering Array Mathematical Notation (OPATHS)

Aminu Aminu Muazu[1]
Department of Mathematics & Computer Science.
Umaru Musa Yar'adua University
Katsina, Nigeria.
aminu.aminu@umyu.edu.ng

Abubakar Aminu Muazu[2]
Department of Mathematics & Computer Science.
Umaru Musa Yar'adua University
Katsina, Nigeria.
abubakar.muazu@umyu.edu.ng

Abstract— *Software testing is a step by step process of employing a product in order to make sure it entertains user's specification requirements. Testing is compulsory, to reassure that software works perfectively and to confirm that a software was successfully tested, the software should be tested exhaustively just to make sure that the software cannot be demolished by some random accidents, but the exhaustive testing is unattainable. There are many existing strategies that minimized the test suite size in a software system, but most of these strategies based on optimization algorithm are using one-test-at-a-time approach and none of them have been adopt one-parameter-at-a-time approach that is based on harmony search algorithm. Therefore, this paper will describe a new strategy called OPATHS. OPATHS is the first strategy based on Harmony Search Algorithm that adopt one-parameter-at-a-time approach. OPATHS was designed only to support Mixed Covering Array notations with a uniform interaction strength and from the result obtained in the experiments gives a comparable result and always appears to be best.*

Index Terms— *Software testing, combinatorial testing, d-way techniques, one parameter at a time approach, Harmony Search Algorithm, Uniform Interaction strength.*

I. INTRODUCTION

In the modern world, software is use in every detail of social life and is becoming the key engine of economic and as well social progress that affects social and economic development. Nowadays, almost our live rely more on the execution of software (accurately), whether the software is a mobile phone application, TV application, computer application, etc. Also in the business, we carry out each day with credit/debit card acquisitions, money transfer, use of internet, e-mail, chatting, and so on.

The entire aspiration of a software company is to ensure that a software is delivered with a high quality to it customers. Therefore, to achieve a high-quality software, the software need to be tested [2]. The software testing makes sure that software achieves the user requirements, such that to avoid failures visible to customers [2]. In software development lifecycle, software testing is giving a higher priority and lack of testing may lead to harmful ends which includes the loss of an important data, or even the lives of people [18].

The major aim of software testing is to minimize the recognized risk of software which is worthless to an admissible value [4]. As such, the software engineers will come up with a large number of test data [4]. If every possible test has been run, the bugs or defects will be detected that is called exhaustive testing, that is to testing all possible combinations of inputs data and execution paths, but this is impossible in real world software, based on the general principles of software testing, it would need undetermined time, and enormous resources [10].

Combinatorial testing is a type testing whereby for each *d*-way (where *d* indicates the combination strength) combination of possible values of parameters of a given system, that every combination of values of these *d* parameters can be covered by at least one test case [15] [14].

All Combinatorial testing approaches are categorized into: "one-test-at-a-time" approach (OTATA) and "one-parameter-at-a-time" approach (OPATA). As for OPATA, the main example is IPOG [12].

There are many existing strategies like GA [7], SA [8], HSS [3] that minimized the test suite size in a system, but the majority of these strategies based on optimization algorithm and are using OTATA but none of them has been

based on optimization algorithm and using OPATA. Even though, IPOG and its family are one of the existing strategy that adopt one-parameter-at-a-time approach, but it is not based optimization algorithm [12].
D-way strategy is referring to an interaction testing, which reduces test data from a software system based on a given interaction strength (*d*) [5].

Therefore, this paper will propose and implement a d-way combinatorial testing strategy which based on harmony search algorithm by adopting OPATA.

II. RELATED WORK

Similar to any other engineering process, software development is subjected to cost. Nowadays, software testing consumes most of the time and cost spent on software development. This cost decrease rapidly as testing time decreases. Most of the time, a software may be released without being tested sufficiently because of marketing pressure as well as the intention to save time and cut costs. As such, many researchers have developed various strategies to solve this problem with the intention to generate near optimal test suite [4].

Majority of the existing strategies on d-way combinatorial testing concentrated on pairwise testing that can detect any fault that occur between the interaction of two parameter's value [12]. Furthermore, existing *d*-way strategies are of two categories, these categories are based on the dominant approaches that are either algebraic approach or computational approach [5] [12] [20].

There are strategies that adopt computational approach, examples of these strategies are: IPOG [12], SA [17], HSS [3] [19] and GA [15].

The existing strategies that support *d*-way interaction can be categorized into two approaches. These are: OTATA and OPATA.

A. OTATA

The main idea of OTATA is straight forward: it generates the test case one by one continuously until the coverage requirement is met. Therefore, during this process, each new test case is covers as many uncovered target combinations as possible, in such a way that the total number of test cases in the test suit can be reduced (minimized) [6]. To generate this individual test case, some algorithms are used (developed) to select a best individual test case at each time. Some of these algorithms are: Greedy algorithms GA [2], AETG [6], Simulated Annealing (SA) [8], ACA [11], and HSS [3] [19].

B. OPATA

The OPATA is quite different compare to OTATA. The main idea of OPATA is straight forward: It begins with selected parameters, then it iteratively adds one parameter till all parameters are covered (i.e. horizontal growth) and new test cases could be added (i.e. vertical growth) to ensure maximum interaction coverage [16]. OPATA ensure that the total number of test cases in the test suit are reduced (minimized). To generate this individual test case, some algorithms are used (developed). Some of these algorithms are: IPO [16], [12], ReqOrder [13], ParaOrder [13].

III. HARMONY SEARCH ALGORITHM

The Harmony Search Algorithm (HSA) was initially proposed by Geem and apply to solve the optimization problem of water distribution networks in 2000 [9]. As a novel population-based meta-heuristic algorithm, during the recent years, it has gained great research success in the area of mechanical engineering, control, signal processing, etc. When musicians compose the harmony, they usually try various combination of the music pitches stored in their memory, which can be considered as an optimization process of adjusting the input (pitches) to obtain the optimal output (perfect harmony). The music improvisation is a process of searching for the better harmony by trying various combinations of pitches that should follow any of the following three rules [9]:

- By playing any pitch from the memory.
- By playing an adjacent pitch of one pitch from the memory.
- By playing a random pitch from the possible range.

The three rules in the HS algorithm are effectively directed using two essential parameters: Harmony Memory Considering Rate (HMCR) and Pitch Adjustment Rate (PAR) [9].

The first step will initialize the HS memory (HM). The initial HM consists of a given number of randomly generated solutions to the optimization problem under consideration [2].
The second step will improvise a new solution from the HM. Each component of this solution is obtained based on the HMCR [2].
The third step will update the HM. The new solution from second step is evaluated. If it yields a better fitness than that of the worst member in the HM, it will replace that one. Otherwise, it is eliminated [2].

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) [2].

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

Fig. 1. The harmony search algorithm [2].

IV. OPATHS STRATEGY

The framework of OPATHS strategy can be describe here that can always construct a minimum test suite. Under the circumstances that different test case construction algorithm that have been developed with an objective to generate a near minimum test suite. OPATHS have been designed based on HSA that have been proposed some modifications to work on OPATA instead of OTATA used in normal HSA in [2]. These modifications will enhance and improve the generation of final test suite to be a near optimum size.

The OPATHS is developed to support uniform interaction strength test suite. This strategy is comprising of two main algorithms as follows:

- Initial pairs algorithm,
- The test suite generation algorithm.

OPATHS will start by after the longest pair has been generated, after then it will follow by these steps below:

Step 1: Initializing the HM

OPATHS will start by initializing the HM with a random test pair by considering the longest pair generated by initial algorithm. Test pairs from the initial pair algorithm will be selected to put a random value from the next parameter. Also, the best test case will be selected based on number of covered and uncovered interactions and then it put in the next pairs. The HM most have a specific size, the size of the HM here OPATHS is five (5).

Step 2: Improvise a new solution from the HM

The OPATHS here will set a value of HMCR to be random value between 0-1 in order to improvise. This improvisation is based on the HMCR value. If the HMCR< 0.7 it will improvise locally, else it will improvise globally. For the local improvisation PAR will be set to be a random value between 0 – 1 in order to make an adjustment for the selected test pair. It will adjust if PAR<0.9, otherwise no adjusting. Adjusting here will change the value of the current selected parameter to another value within the current selected parameter.

Step 3: Updating the HM

From step 2 above, a new solution is evaluated (a new pair). So, if that solution is yields a better fitness than that of the worst member in the HM, it will replace that one. Otherwise, it is eliminated.

Step 4: Iteration

This step will repeat step 2 and step 3 until a maximal number of iterations is met and it generate the near optimal final test suite. Here in OPATHS, number of iteration is ten (10) times.

```
Start
Declare LongestPair list= null, NextPair list= null, FinalTest list= null
Read file
Sort the Parameters in respect to their values
Select & generate parameter combination based on d
Generate LongestPair
Loop1//LOOP ALL PARAMETERS TO PICK EACH
   Loop2 //LOOP LONGESTPAIR TO PICK EACH PAIR
        If ( NextPair list is not empty)
        Select a pair from LongestPair and Put into HM //  five times
              Loop3
                    Select each pair from the HM &  Measure the weight
               Loop3 end
               if (weight== 0)
                     Add selected pair from HM to nextpair list
                Else // improvise. Ten (10) times
                     Loop4
                               Initialize HMCR (ranges 0-1)
                               Declare newPair
                               If (HMCR<0.7)    //Do local improvisation
                                  Initialize PAR (ranges 0-1)
                                   If (PAR<0.9)  //adjust
                               newPair = Change the value of last parameter of  the selected test pair from
                                 HM within the same parameter
                                 else  //no adjust
                                  newPair=selected pair from HM
                          else        // Do global improvisation
                          newPair=select random test pair from HM
                                    Measure newPair weight
                                    If (newPair weight == 0)
                                            Add newpair to nextpair list
                                     Else if (newPair weight >= worst)   //UPDATE HM
                                         Replace newPair with the worst pair in HM
                           Loop4 End
                           Add best pair in hm to nextpair list
        Else
              Add a random pair to nextpair list
     Loop2 End
     LongestPair list = nextPair list
Loop1 End
FinalTest list = LongestPair
Check Missing Pair
If (missing pair == true)
    Update FinalTest list
Else
     Ignore missing pair
// end of check missing pair
Return FinalTest list
End
```

Fig. 2. The OPATHS pseudo code

V. ANALYSIS AND PERFORMANCE

Evaluation of the OPATHS focuses only on one main criteria: it efficiency/performance to generate better test suites sizes compared with existing strategies.

The OPATHS have adopted the following parameter settings (see Table I), in order to take the best result from the ten (10) runs improvisation:

TABLE I: OPATHS PARAMETER SETTINGS

S/N	Parameter	Value
1	HMS	5
2	HMCR	0.7
3	PAR	0.9
4	Improvisations/iterations	10

The Table I above describes the parameters settings used in OPATHS. The Harmony memory size (HMS) is set to five (5), that means it can only accommodate five test pairs at a time. The HMCR is set 0.7 to give permission for either to do local or global improvisation, when it's higher than 0.7 it will do local improvisation otherwise global. The PAR is set to 0.9 to give permission to adjust a value or not, that is when it's higher than 0.9 it will adjust a parameter value otherwise no adjustment. The final parameter setting is iterations, the iteration count the number of improvisation which is set to 10.

Based on the above-mentioned criteria, the following sub-sections present the complete evaluation:

Comparison of OPATHS with other strategies

To benchmark OPATHS against other strategies, OPATHS is compared with other available strategies, including HSS, SA, GA, ACA, AETG, IPOG, Jenny, TVG, and PSTG. Here, the comparison aims to investigate the OPATHS generated test suite size against other strategies based on well-known benchmark configurations. Hence, OPATHS results are directly compared with published results for strategies in [2].

A number of system configurations is divided into fourteen groups in order to compare the performance of OPATHS against other strategies [2]. For the comparative purposes, an experiment is adopted on each of the fourteen configuration ten times, this is because there is a random selection of values of parameter which may have different test suite size on each experiment. In the experiment, only a system configuration with covering array notation and mixed covering array notation are adopted with a uniform interaction strength. The configurations are shown as follows:

S1= CA (N; 2, 3^4), **S2**= CA (N; 2, 3^{13}),
S3= CA (N; 2, 10^{10}), **S4**= CA (N; 2, 15^{10}),
S5= CA (N; 2, 5^{10}), **S6**= CA (N; 3, 3^6),
S7= CA (N; 3, 4^6), **S8**= CA (N; 3, 5^6),
S9= CA (N; 3, 6^6), **S10**= CA (N; 3, 5^7),
S11= MCA (N; 2, $5^1 3^8 2^2$),
S12= MCA (N; 2, $7^1 6^1 5^1 4^6 3^8 2^3$),
S13= MCA (N; 3, $5^2 4^2 3^2$),
S14= MCA (N; 3, $10^1 6^2 4^3 3^1$)

TABLE II: COMPARISON IN TERMS OF TEST SUITE SIZE FOR DIFFERENT CONFIGURATION (WHEN $2 \leq d \geq 3$)

Configuration	HSS	SA	GA	ACA	AETG	IPOG	Jenny	TVG	PSTG	OPATHS
S1	9	9	9	9	9	9	10	11	9	9
S2	18	16	17	17	17	20	20	19	17	16
S3	155	NA	157	159	NA	176	157	208	NA	228
S4	341	NA	NA	NA	NA	373	336	473	NA	559
S5	43	NA	NA	NA	NA	50	45	51	45	45
S6	39	33	33	33	38	53	51	49	42	32
S7	70	64	64	64	77	64	112	123	102	88
S8	199	152	125	125	194	216	215	234	NA	177
S9	336	300	331	330	330	382	373	407	338	325
S10	236	201	218	218	218	274	236	271	229	202
S11	20	15	15	16	20	19	23	22	NA	15
S12	48	42	42	42	44	43	50	51	48	42
S13	119	100	108	106	114	111	131	136	NA	102
S14	378	360	360	361	377	383	399	414	385	360

The highlighted (grey) cells in Table II above show the smallest (best) generated size of the test suite from each strategy, the highlighted (yellow) cells show the second best generated size, and the (NA) cells refers to not available, meaning that the strategies' results are not reported in their respective publications.

Based on the results shown in Tables II it is clear that OPATHS performance is affected by the increasing number of V and P and also better when the configuration system is a mixed covering array. The performance of the mixed covering array has a better performance than the covering array configuration. In fact, it can be seen that

OPATHS outperforms all other strategies in most cases considered, because most of the time it appears to be the best or second best.

Due to the fact that OPATHS can support the mathematical notation CA and MCA (uniform interaction strength) and from the result obtained in the experiments (Table: II) appears that OPATHS is always best at configurations with MCA notations compare to the CA notations which is not perfect.

Therefore, it can be concluded that OPATHS is useful for supporting software testing.

VI. CONCLUSION

In this paper, we have described an innovative approach of applying OPATA called OPATHS as a strategy for *d*-way test generation. Comparatively, the performances of OPATHS with some existing strategies have been encouraging.

To the best of my knowledge, OPATHS is the first Harmony Search based strategy supporting OPATA which addresses the problem of *d*-way test suite generation. The main features of OPATHS is that it's well optimized and has been exceptional in performance.

Finally, OPATHS is still in a prototype form, an obvious starting point for future work would be to complete the implementation. For example we would personally suggest the following recommendations: To support a VCA notation with variable interaction strength, and to support IOR notation with IO based relation as well.

REFERENCES

[1] M. Kaur, and R. Singh, "A review of software testing techniques," International Journal of Electronic and Electrical Engineering, 7(5), 463-474, 2014.

[2] A. A. Alsewari, and K. Z. Zamli, "Interaction Test Data Generation Using Harmony Search Algorithm," Symposium on industrial electronics and applications, Page 559-564, 2011.

[3] 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.

[4] 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.

[5] A. A. Alsewari, K. Z. Zamli, and B. AL-Kazemi, "Generating t-way test suite in the presence of constraints," Journal of Engineering and Technology, 6(2), 2180-3811, 2015.

[6] 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, 303-309, 1994.

[7] P. Ranjan, and T. Kim, "Application of genetic algorithm in software testing," International Journal of Software Engineering and Its Applications, 3(4), 2009.

[8] X. Deng, Z. Wen, Y. Wang, and P. Xiang, "An improved PSO algorithm based on mutation operator and simulated annealing," International Journal of Multimedia and Ubiquitous Engineering, 10(10), 369-380, 2015.

[9] W. Geem, H. Kim, and V. Loganathan, "A new heuristic optimization algorithm: harmony search simulation," International Journal of Computer science and software engineering, 76 (2), 60–68, 2001.

[10] B. Hambling, "Software testing an ISTQB–ISEB foundation guide," 2011.

[11] I. Hassan, and Z. Kamran, "Using Ants as a Genetic Crossover Operator in GLS to Solve STSP," International Conference of Soft Computing and Pattern Recognition, Vol1, 2014.

[12] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: A general strategy for t-way software testing," 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 549-556, 2007.

[13] Z. Wang, C. Nie, and B. Xu, "Generating Combinatorial Test Suite for Interaction Relationship," Journal of Software ACM, ISBN 978-1-59593-724, 2007.

[14] R. Othman, and K. Z. Zamli, "T-way strategies and its applications for combinatorial testing," International Journal on New Computer Architectures and Their Applications (IJNCAA), 1(2), 459-473, 2011.

[15] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using artificial life techniques to generate test cases for combinatorial testing," 28th Annual International Computer Software and Applications Conference IEEE, 1, 72-77, 2004.

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

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

[18] M. I. Younis, and K. Z. Zamli, "MC-MIPOG: A parallel t-way test generation strategy for multicore systems," ETRI Journal, 32(1), 73-83, 2010.

[19] K. Z. Zamli, Y. A. Basem, and K. Graham, "A Tabu Search hyper-heuristic strategy for t-way test suite generation," Journal of Applied Soft Computing, 44, 57-74, 2016.

[20] 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., 2011.