A Pairwise Test Suite Generator Based on Melody Search Algorithm

Toh S. Yuen, Ala’a A. Al-Omoush, Foo W. Wen, Goh G. Hau, 
Abdul Rahman and A. Alsewari
Department of Software Engineering, Faculty of Computer Systems and 
Software Engineering, Universiti Malaysia Pahang, Gambang, Malaysia

Abstract: Melody Search Algorithm (MS) has been adopted to solve many optimization problems such as 
combinatorial testing. This paper aims to introduce MS as a pairwise testing strategy called a Pairwise Test suite 
generator based Melody Search Algorithm (PTMS). A pairwise testing is an operative approach in the 
combinatorial test suite construction. It will minimize the constructed test suite size to save the testing time with 
effective defects being detected. The proposed strategy generates the test suite with real-valued variables. A 
comparative evaluation revealed that the PTMS is efficient in constructing a minimum test suite with the 
existing strategies.

Key words: Pairwise testing, Melody Search (MS) algorithm, test data generation, combinatorial testing, 
variables

INTRODUCTION

In this information technology era, there is a huge 
influence of high technology and artificial intelligence 
when creating new software products. This new method 
provides an effective way of bringing high-quality 
software products to the end user. Many fields also rely 
on this method, especially in the Research and 
Development (R and D) area. Evidently, many manual 
processes are being performed by certain software 
products or artificial intelligence. Basically, every 
product created is operated by the combination of 
hardware and software to implement each feature 
(Perrouin et al., 2012). There is a close relationship 
between hardware and software, both play important roles 
to avoid failure.

Failure of products will occur when a human action 
produces some error or bug in the software and this will 
lead to the defects which will cause a malfunction when 
exeucted. This problem can cause serious damage to 
system function, loss of time especially for a critical 
system and higher cost of maintenance. Therefore, 
software testing takes first priority in any Software 
Development Life Cycle (SDLC) in order to ensure high 
quality software and to minimize the occurrence of failure 
of the software.

Software testing is defined as the process of 
exeucting a program on finding possible errors and 
validating the software or system against its specification 
(Myers et al., 1979). It is known from the studies of seven 
principles that exhaustive testing is impossible to execute 
all the test cases for a real software product (Wang et al., 
2013). A complete testing is impossible because there are 
many possible combinations of inputs and pre-condition 
test case for software.

Pairwise testing is an effective combinatorial method 
of software testing used to minimize the number of test 
that is needed for input parameters to a system and 
the interactions between two input parameters values 
(McCaffrey, 2010). This strategy generates test cases that 
cover all the possible input combinations in order to 
include the test data and to reduce the possibilities of 
faults due to interaction (Perrouin et al., 2012). There are 
many pairwise testing strategies that are available such as 
Simulated Annealing (SA) (Cohen et al., 2007), Automatic 
Efficient Test Generator (AETG) (Cohen et al., 1997), 
Genetic Algorithm (GA) (Flores and Cheon, 2011; 
McCaffrey, 2010) Ant Colony Algorithm (ACA) 
(Shiba et al., 2004), Bat Pairwise Test Strategy (BPTS) 
(Alsari et al., 2015), Harmony Search Strategy (PHSS) 
(Alsewari and Zamli, 2012), In-Parameter-Order (IPO) 
(Lei et al., 2007, 2008) and Intersection Residual Pair Set 
(IRPS) (Younis et al., 2008, 2009). Although, the listed 
strategies are useful, none of them can give assurance in 
producing optimum results for every case study. It 
should be noted that some of the existing pairwise 
testing strategies are based on optimization algorithms 
such as GA, HSS, ACA, BPTS and SA. Therefore,

Corresponding Author: Toh S. Yuen, Department of Software Engineering, Faculty of Computer Systems and 
Software Engineering, Universiti Malaysia Pahang, Gambang, Malaysia
this study will introduce a new pairwise testing strategy based on a new optimization algorithm called Melody Search Algorithm.

**Problem statement:** The main outcome of the software testing is the determination of defects for the existing software product (Myers et al., 1979). Table 1 shows the features on a laptop which consist of different Operating System (OS), Processors, System Type, Random Access Memory (RAM), Graphics Card, Battery, Universal Serial Bus (USB), Hard Drives, Screen Resolutions and Keyboard. From Table 1, there are 10 parameters where each parameter has two values. Therefore, exhaustive testing which is equivalent to $2 \times 2 \times 2 \times 2 \times 2 \times 2 = 1024$ possible combinations need to be tested in order to cover all the test cases. If one test case needs 5 minutes to be tested, the total minutes to complete the test will be 5120 minutes or approximately using 3 days to complete all test cases. In practice, it is impossible to test all combinations due to the time and cost constraints (Wang et al., 2013). Therefore, it is necessary to minimize the test suite without missing any of the system’s parameters combinations based on each of the two parameters (McCaffrey, 2010). Subsequently, the test suite that can be produced by a pairwise strategy will contain 10 test cases instead of 1024 test cases.

**Existing pairwise strategies:** The surveys in this study will be based on existing research on pairwise testing. The existing pairwise testing strategies can be differentiated into two approaches namely, one test at a time (OTAT), and one parameter at a time (OPAT).

**One test at a time (OTAT):** OTAT is an approach that uses several existing pairwise testing strategies. OTAT is meant to generate a test case for each time iteration is performed. The rule of OTAT involves the generation of entire test cases one after the other until every test cases have been covered by all possible interaction. There are many existing strategies in this approach. Some of these strategies adopt the computational approach such as such as AETG and its family AETG2 and mAETG SAT (Cohen et al., 1997), AllPairs (Bach, 2001), 02Way (Klaub et al., 2008), Pairwise Independent Combinatorial Testing (PICT) (Czerwonska, 2006) while others adopt the optimization algorithms approach such as SA and its variant SA SAT (Cohen et al., 2007), GA (Flores and Cheon, 2011; McCaffrey, 2010), ACA (Shiba et al., 2004), LAHC (Zamli et al., 2015), PHSS (Alsewari and Zamli, 2012) and BPTS (Alsariea et al., 2015).

**One parameter at a time (OPAT):** OPAT is another approach used in generating all test cases by starting from the first pair after which each parameter’s values is added to every iteration performed until all pairs are covered. In this way, we have selected one of the pure computational techniques which is IPO (Lei et al., 2007, 2008) and one of the improvements of computational techniques like IRPS (Younis et al., 2008, 2009).

From the above, it is known that most of the OTAT approach strategies are classified as a non-deterministic approach since the generated test cases cannot be determined. As such, the same input parameter may appear in the different test cases for OTAT. Unlike the OTAT, OPAT approach strategies are classified as deterministic approaches which are always generating same test cases result by using a specific input parameter.

From the aforementioned strategies and from one of the seven testing principles, there is a need to generate different test cases in order to increase the percentage of defects detection. Based on that, this research will introduce a new non-deterministic test cases generation strategy based on Optimization algorithm called Melody Search algorithm (MS).

MS has been successfully applied to solve many optimizations problems (Ashrafi and Dariane, 2011, 2013). MS is an advanced improved version of Harmony Search algorithm. MS process is like HS process but MS has a number of memories while HS has only single memory. Furthermore, MS has two type of improvisation which will be discussed in the next section.

**MATERIALS AND METHODS**

**Pairwise test cases melody search strategy:** This research is to develop a pairwise test case generation strategy based on Melody Search Algorithm (PTMS). This study is going to discuss the flow of framework, design approach, implementations of PTMS that are involved in this research.
Melody search algorithm (MS) for pairwise test strategy:
Melody Search Algorithm is an improved optimization algorithm from Harmony Search Algorithm (HS) which is inspired by its basic concepts (Ashrafi and Dariane, 2011, 2013). This algorithm is derived from musical interactions within a group of musicians with different style and ideas to find the best version of melodies in a horizontal line which consists of a series of pitches. Every member from the group of the musicians will perform a smooth and nice melody consisting of series of pitches and store into their own Player Memory (PM). The structure of MS Algorithm is quite different from HS Algorithm, although the basic concepts of HS which are based on music improvisation process are applied. A melody memory will be made up from several PM which is owned by every musician and each will be compared to Harmony Memory (HM) which has only a single HM. The main steps of PTMS strategy can be described as following:

**Step 1 (initializing of problem):** This step will initialize the optimization problem and adopts several parameters that need to be determined. The parameters that need to be determined include the number of parameters, number of values for each parameter, interaction tuples based on each two parameters (IT), Final Test List (FTL) number of player memories (PMN), Player Memory Size (PMS), maximum number of Iterations for Initial/first phase (NIT), Band Width (bw), Player Memory Considering Rate (PMCR) and Pitch Adjusting Rate (PAR) (Fig. 1-3).

**Step 2 (initial/first phase):** Based on the PMS size, initializing of PM with random melodies (test cases).

**Step 3 (improvisation 1):** Then improvising a new melody from each PM. The new melodic line will be generated based on PMCR, PAR or randomization.

![Flow chart of step 1 initialization of problem](image-url)
Initialize every Player Memory (PM) by random generate test case
based on interaction list generated
Evaluate coverage of test case (number of interaction pairs covered)
All PM filled?
i=0, i < 10 (No.of iterations), i++
Random generate PMCR, saveranPMCR
Random pick a test case from PM
Random generate PAR, saveranPAR
Improvise a new test case by shifting test case to its neighbouring values with specified bandwidth (±1)
Evaluate coverage of test case (number of interaction pairs covered)

Fig. 2: Flowchart of step 2 initial/first phase

Update coverage of every test case in PM
Select best test case with highest coverage from both PM
Improvise a new test case through differences of values between the two test cases selected
Evaluate coverage of test case (number of interaction pairs covered)
Coverage > Least coverage of test case in PM?
No
Select & Add the best test case with most coverage into Final Test List
No of uncover interaction pairs left != 0 ?
Print test cases of Final Test List
End

Coverage < Least coverage of test case in PM?
Yes
Update & Replace the least coverage test case from PM
No of uncover interaction pairs left > 0 ?
Random generate test case to fill up the place of best test case removed in PM
Update coverage of every test case in PM
Step 2

Fig. 3: Repeating the improvisation and the updating steps until the iteration number reach the maximum number of NII

Updating every test case in each PM. The existing test case with worst fitness function value will be replaced if the new melody has a better fitness function value.
Step 4 (improvisation II): A possible range of best arrangement of pitches in a test case which can be varied through iterations is initiated between music players. Improvising a new test case from each PM according to the range of pitches determined. The existing test case with worst fitness function value in each PM will be replaced by the new test case if the new test case has a better fitness function value. Identify any possible range of pitches in melody for next improvisation.

Step 5 (add to final test list): The best test case overall PMs will be added to FTL. Then step 2 to step 5 will be repeated until all interaction tuples in it.

RESULTS AND DISCUSSION

The performance of the proposed strategy is discussed in terms of generating and minimizing the test cases list size.

Two experiments were conducted to evaluate the performance of PTMS. The proposed strategy was conducted in the environment of Windows 7 Home Premium 64-bits Operating System, Laptop with Intel Core i5, 2.5GHz CPU, 4.00GB RAM and the application of NetBeans IDE 8.0.2, Oracle JDK 8.0 and JRE 8.0 installed.

The various attributes that define PTMS were applied in every experiment. The number of PMN is two, PMS is ten for each player memory, NI for improvisation step is ten, bw for pitch adjustment is ±1, PMCR is 0.9 and PAR is 0.2. Results in terms of test list size generated are derived from average test list size of 20 times execution. The results of comparing strategies are expressed as the average result of the 20 times execution while the results of PTMS are shown in both the average result (row 1 = r1) and the best result (row 2 = r2).

Experiment 1: The data specifications and the result for existing strategies are collected from publication by Alsariera et al. (2015). There are 6 case studies with different input data specifications involved in this experiment as shown in Table 2. The results of experiment 1 are presented in Table 3.

Based on Table 3, PTMS generated the most optimized result of 9 test cases in S1 which is similar to Jenny and IPRS but better than 10 test cases generated in All-Pairs and G2Way. In S2, PTMS generated an optimized result with 10 test cases which is similar to All-Pairs and G2Way. Its result is better than Jenny with 13 test case and AETG2 with 11 test cases. Most of the tools might generate better result than PTMS which are 9 and 10 test cases. However, it can be proved that all the strategies successfully generated satisfactory tests listed in S2.

<table>
<thead>
<tr>
<th>Case study</th>
<th>Covering array representation</th>
<th>Input specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>CA (N, 2, 3)</td>
<td>Three 3-valued parameters</td>
</tr>
<tr>
<td>S2</td>
<td>CA (N, 2, 3)</td>
<td>Three 3-valued parameters</td>
</tr>
<tr>
<td>S3</td>
<td>CA (N, 2, 3)</td>
<td>Three 3-valued parameters</td>
</tr>
<tr>
<td>S4</td>
<td>CA (N, 2, 100)</td>
<td>Three 10-valued parameters</td>
</tr>
<tr>
<td>S5</td>
<td>CA (N, 2, 100)</td>
<td>Three 10-valued parameters</td>
</tr>
<tr>
<td>S6</td>
<td>CA (N, 2, 50)</td>
<td>Three 5-valued parameters</td>
</tr>
</tbody>
</table>

Table 3: Experiment 1 results in terms of test list size generated

<table>
<thead>
<tr>
<th>Strategies</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
</tr>
</thead>
<tbody>
<tr>
<td>AETG</td>
<td>NA</td>
<td>9</td>
<td>15</td>
<td>NA</td>
<td>180</td>
<td>NA</td>
</tr>
<tr>
<td>AETG2</td>
<td>NA</td>
<td>11</td>
<td>17</td>
<td>NA</td>
<td>198</td>
<td>NA</td>
</tr>
<tr>
<td>IPO</td>
<td>NA</td>
<td>9</td>
<td>17</td>
<td>169</td>
<td>212</td>
<td>47</td>
</tr>
<tr>
<td>SA</td>
<td>NA</td>
<td>9</td>
<td>16</td>
<td>NA</td>
<td>183</td>
<td>NA</td>
</tr>
<tr>
<td>GA</td>
<td>NA</td>
<td>9</td>
<td>17</td>
<td>157</td>
<td>227</td>
<td>NA</td>
</tr>
<tr>
<td>ACA</td>
<td>NA</td>
<td>9</td>
<td>17</td>
<td>159</td>
<td>225</td>
<td>NA</td>
</tr>
<tr>
<td>BPTS</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>162</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>PHSS</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>155</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>All-Pairs</td>
<td>10</td>
<td>10</td>
<td>22</td>
<td>177</td>
<td>230</td>
<td>49</td>
</tr>
<tr>
<td>G2Way</td>
<td>10</td>
<td>10</td>
<td>19</td>
<td>160</td>
<td>200</td>
<td>46</td>
</tr>
<tr>
<td>Jenny</td>
<td>9</td>
<td>13</td>
<td>20</td>
<td>157</td>
<td>194</td>
<td>45</td>
</tr>
<tr>
<td>IPRS</td>
<td>9</td>
<td>9</td>
<td>17</td>
<td>149</td>
<td>210</td>
<td>45</td>
</tr>
<tr>
<td>PTMS r</td>
<td>1.93</td>
<td>11.2</td>
<td>23.2</td>
<td>259.40</td>
<td>361.10</td>
<td>59.70</td>
</tr>
<tr>
<td>r2</td>
<td>9</td>
<td>10</td>
<td>22</td>
<td>254</td>
<td>357</td>
<td>58</td>
</tr>
</tbody>
</table>

Experiment 2: The data specifications and the result for existing strategies are collected from publications (Zamli et al., 2015; Al-Sewari et al., 2014; Longshu and Yung, 2012). The conditions for experiment 2 are as follows.

There are 5 case studies with different input data specifications involved in this experiment as shown in Table 4. The results from experiment 2 are presented in Table 5.

Based on Table 5, PTMS generates the most minimum test cases such as 9 test cases in S7 which are similar to most of the other strategies and it is better than PICT which generates 10 test cases. S8, PTMS also generates the most minimum test cases like the other strategies except PICT. S9, S10, S11 and PTMS generate acceptable test cases as compared to the existing strategies results.

<table>
<thead>
<tr>
<th>Case studies</th>
<th>Covering array representation</th>
<th>Input specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>CA (N, 2, 3)</td>
<td>Three 3-valued parameters</td>
</tr>
<tr>
<td>S2</td>
<td>CA (N, 2, 3)</td>
<td>Three 3-valued parameters</td>
</tr>
<tr>
<td>S3</td>
<td>CA (N, 2, 3)</td>
<td>Three 3-valued parameters</td>
</tr>
<tr>
<td>S4</td>
<td>CA (N, 2, 100)</td>
<td>Three 10-valued parameters</td>
</tr>
<tr>
<td>S5</td>
<td>CA (N, 2, 100)</td>
<td>Three 10-valued parameters</td>
</tr>
<tr>
<td>S6</td>
<td>CA (N, 2, 50)</td>
<td>Three 5-valued parameters</td>
</tr>
</tbody>
</table>

It was observed that some strategies did not show certain case studies. For instance, strategies AETG, AETG2, IPO, SA, GA and ACA did not display any results of S1 with three 3-valued parameters while strategies AETG, AETG2, SA, GA and ACA did not reveal the results for S6 with 10 5-valued parameters.
Table 5: Experiment 2 results in terms of test list size generated

<table>
<thead>
<tr>
<th>Strategies</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA-SAT</td>
<td>9</td>
<td>16</td>
<td>25</td>
<td>36</td>
<td>49</td>
</tr>
<tr>
<td>mAETG-SAT</td>
<td>9</td>
<td>16</td>
<td>25</td>
<td>37</td>
<td>52</td>
</tr>
<tr>
<td>PICT</td>
<td>10</td>
<td>17</td>
<td>26</td>
<td>39</td>
<td>55</td>
</tr>
<tr>
<td>TestCover</td>
<td>9</td>
<td>16</td>
<td>25</td>
<td>36</td>
<td>49</td>
</tr>
<tr>
<td>LAHC</td>
<td>9</td>
<td>16</td>
<td>25</td>
<td>38</td>
<td>51</td>
</tr>
<tr>
<td>PHSS</td>
<td>9</td>
<td>16</td>
<td>25</td>
<td>36</td>
<td>49</td>
</tr>
<tr>
<td>BPTS</td>
<td>9</td>
<td>16</td>
<td>25</td>
<td>36</td>
<td>49</td>
</tr>
<tr>
<td>PTMS 1</td>
<td>9</td>
<td>18</td>
<td>29.5</td>
<td>43.7</td>
<td>59.4</td>
</tr>
<tr>
<td>@2</td>
<td>9</td>
<td>16</td>
<td>29</td>
<td>42</td>
<td>57</td>
</tr>
</tbody>
</table>

Based on the overall result generated in experiment 1 and experiment 2, the overall view shows that results of PTMS are better than some of the existing strategies' results but the best results are not shown sometimes. Although some of the strategies might generate better results than PTMS in overall cases, the PTMS successfully generate a satisfactory test list size.

CONCLUSION

This study illustrated the pairwise testing and its different approaches. The capability of adopting the Melody Search algorithm in pairwise testing has been elaborated. PTMS is the first pairwise testing strategy implemented in the Melody Search algorithm to produce a test cases with minimum size. Furthermore, some experiments have been carried out to evaluate and prove the effectiveness of PTMS. Future enhancement of PTMS should be developed to support the constraints. While there are many optimization algorithms which have been raised to solve optimization problems, the research in the test cases generation can adopt one of these algorithms.

ACKNOWLEDGEMENTS

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

REFERENCES


