

Pairwise Testing Tools Based On Hill Climbing Algorithm (PTCA)

By

LIM SENG KEE

**Thesis submitted in fulfilment of the requirements
For the degree of
BCS (Software Engineering with Honours)**

Dec 2014

PERPUSTAKAAN UMP



0000103250

Pairwise Testing Tools Based On Hill Climbing Algorithm (PTCA)

LIM SENG KEE

BACHELOR OF COMPUTER SCIENCE
(SOFTWARE ENGINEERING WITH
HONOURS)

University Malaysia Pahang

SUPERVISOR'S DECLARATION

We hereby declare that we have checked this thesis and in our opinion, this thesis is adequate in terms of scope and quality for the award of the degree of Bachelor of Computer Science (Software Engineering with Honours)

Signature

: 

Name of main supervisor

: Professor KAMAL ZUHAIRI BIN ZAMLI

Position

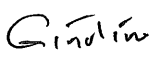
: Professor

Date

: 23/12/2014

STUDENT'S DECLARATION

I hereby declare that the work in this thesis is my own except for quotations and summaries which have been duly acknowledged. The thesis has not been accepted for any degree and is not concurrently submitted for award of other degree.

Signature : 
Name : LIM SENG KEE
ID Number : CB11049
Date : 28/12/2014

DEDICATION

I would like to dedicate my thesis to my beloved supervisor, friends and family who supported me each step of the way.

ACKNOWLEDGEMENT

I would like to take this opportunity with great pleasure to acknowledge and extend my heartfelt gratitude to the following persons who have made the successful completion of this final year project possible.

First and foremost, my deepest gratitude goes to my supervisor, Dr Kamal Zuhairi Zamli, for his advice, guidance and patience throughout the process until the completion of the project. Under his supervision, I am well trained to work independently. Thanks for his endless vital encouragement and support given.

Last but not least, I am appreciated with the mentally support from my family and friends even they are not directly contributed to my project.

ABSTRACT

Failure in software can cause serious damage and because of this it is essential to test the software thoroughly. Software testing is the activity to find the defects in the components or systems. The main problem in software testing is to test the system exhaustively as it is impossible to do so. To reduce the test cases generated during testing process, combinatorial algorithms which consisting Hill Climbing algorithm and T Way Combination algorithm as described in twayGenerator (Kamal Zuhairi Zamli, 2007) have been studied and reviewed. The actual implementation of the algorithm which is in java programming language, the program is implemented on Net Bean 7.0.1. Assumptions have been made for the program. Several experiments were conducted included demonstrating the correctness of the algorithms, demonstrate PTCA successful in reducing test suite size, and benchmarking PTCA against existing strategies in order to prove PTCA achieved its objectives. Based on the experimental results, PTCA has successful in reducing the test size.

\.

ABSTRAK

Kegagalan dalam perisian boleh menyebabkan kerosakan yang serius dan ini menyebabkan untuk menguji perisian dengan sempurna adalah penting. Pengujian perisian adalah aktiviti untuk mencari kecacatan dalam komponen atau sistem. Masalah utama dalam menguji perisian adalah untuk menguji sistem secara menyeluruh kerana untuk berbuat demikian adalah mustahil. Untuk mengatasi masalah tersebut, satu kajian yang menggabungkan Algorithma Pendalaan Bulat dengan Pengujian t-hala telah dijalankan. Kajian ini, dilaksanakan dalam Bahasa programing java dengan menggunakan Net Bean 7.0.1, sebagai alat perlaksanaa. Selepas kajian ini lengkap, beberapa percubaan telah dijalankan untuk memastikan kajian ini mencapai objektif-objektif yang telah ditentu dalam kajian ini. Keputusan percubaan tersebut telah butikan bahawa PTCA mencapai objektif yang telah ditentung pada awal kajian, iaitu mengurangkan saiz test cases.

TABLE OF CONTENTS

SUPERVISOR'S DECLARATION	I
STUDENT'S DECLARATION	II
DEDICATION	III
ACKNOWLEDGEMENT	IV
ABSTRACT	V
ABSTRAK	VI
TABLE OF CONTENTS	VII
LIST OF FIGURES	IX
LIST OF TABLES	XI
Chapter 1 INTRODUCTION	1
1.1 Software Testing	1
1.2 Problem Statement	2
1.3 Objective	3
1.4 Research Scope	3
1.5 Thesis Organization	3
1.6 Summary	4
Chapter 2 LITERATURE REVIEW	5
2.1 Theoretical Background	5
2.1.1 General Exhausting Testing Approach	5
2.1.2 Equivalence Partitioning Approach	7
2.2 Survey of Existing Pairwise Strategies	8
2.2.1 G2Way	8
2.2.2 Genetic Algorithm (GA)	10
2.3 Summary	11
Chapter 3 METHODOLOGY	12
3.1 Methodology	12
3.2 Hardware & Software	14
3.3 Summary	14
Chapter 4 DESIGN	15
3.1 Design Consideration	15
3.2 Development of PTCA Strategy	15

3.3	Hill Climbing Algorithm	16
3.4	Weight Counting Method	16
3.5	Pairwise Testing Strategy	16
3.6	Development Strategy	17
3.7	Summary	17
Chapter 5 IMPLEMENTATION		18
5.1	Implementation of Exhaustive Testing	18
5.2	Implementation of Pairwise Test Strategy	22
5.3	Implementation of Hill Climbing Algorithm	24
Chapter 6 RESULT AND DISCUSSION		25
6.1	Demonstration of Correctness	25
6.2	Demonstration Of Reducing Test Suite Size	30
6.3	Results of PTCA Benchmarking	32
6.4	Summary	39
Chapter 7 CONCLUSION		40
7.1	Conclusion	40
7.2	Future Work	41
REFERENCES		42
APPENDICES		44

LIST OF FIGURES

<i>Figure 2.1: Algorithm for pair generation for G2Way (Klaib et al, 2008)</i>	8
<i>Figure 2.2: Example of index search (Klaib et al., 2008)</i>	8
<i>Figure 2.3: Example of row index (Klaib et al., 2008)</i>	9
<i>Figure 2.4: :Outline of the proposed GA (Shiba et al., 1997)</i>	10
<i>Figure 3.1: Overview flow of methodology</i>	12
<i>Figure 4.1: Overall design of PTCA</i>	15
<i>Figure 4.2 : Algorithm of hill climbing algorithm</i>	16
<i>Figure 4.3: Algorithm of pairwise strategy</i>	16
<i>Figure 4.4: Algorithm of the system</i>	17
<i>Figure 5.1: 4 inputs with 3 valued</i>	18
<i>Figure 5.2: The possible combination of the input (1)</i>	19
<i>Figure 5.3 : The possible combination of the input (2)</i>	20
<i>Figure 5.4: The possible combination of the input (3)</i>	21
<i>Figure 5.5: The list when the input converting into pairwise (1)</i>	22
<i>Figure 5.6: The list when the input converting into pairwise (2)</i>	23
<i>Figure 5.7: The result of the test</i>	24
<i>Figure 5.8: The common result</i>	24
<i>Figure 6.1: result screen on PTCA</i>	26
<i>Figure 6.2: result screen in PTCA</i>	29
<i>Figure 6.3: Result of S1</i>	34
<i>Figure 6.4: Result of S2</i>	35
<i>Figure 6.5: Result of S4</i>	36
<i>Figure 6.6: Result of S8</i>	37
<i>Figure 7.1: Gantt Chart</i>	44
<i>Figure 7.2: List of pairwise generated (13 parameter with 3 value)</i>	45
<i>Figure 7.3: continue</i>	46
<i>Figure 7.4: continue</i>	47
<i>Figure 7.5:continue</i>	48
<i>Figure 7.6: continue</i>	49
<i>Figure 7.7: continue</i>	50
<i>Figure 7.8 : continue</i>	51
<i>Figure 7.9 : list of possible combination (13 parameter with 3 value)</i>	52

<i>Figure 7.10: continue</i>	<i>53</i>
<i>Figure 7.11 : the result (13 parameter with 3 value).....</i>	<i>54</i>
<i>Figure 7.12: failure print screen during extreme huge data set.....</i>	<i>54</i>
<i>Figure 7.13: Gantt chart</i>	<i>55</i>

LIST OF TABLES

<i>Table 2.1: Example of exhaustive approach.</i>	5
<i>Table 2.2: Possible input of example.</i>	5
<i>Table 2.3: Number of test cases generated by exhaustive approach.</i>	6
<i>Table 2.4: Test cases implement for Equivalence partitioning approach.</i>	7
<i>Table 3.1: List of software used</i>	14
<i>Table 3.2 : List of Hardware used</i>	14
<i>Table 6.1: 4 input with 3 valued</i>	25
<i>Table 6.2 : Symbol of each input.</i>	26
<i>Table 6.3: Result</i>	27
<i>Table 6.4: List of total interaction pairs</i>	27
<i>Table 6.5: 3 parameter with each value of 2</i>	29
<i>Table 6.6: Result</i>	29
<i>Table 6.7 : List of total interaction pairs</i>	30
<i>Table 6.8: Input specification for test A, B, C, D, and F.</i>	31
<i>Table 6.9: Results of test suite size generated with number of runs</i>	31
<i>Table 6.10: Differences between exhaustive testing test suite size and PTT test suite size</i>	32
<i>Table 6.11: Input specification for 8 tests.</i>	32
<i>Table 6.12: Result of the comparison data in term of the test size (Klaib et al., 2008).</i>	33
<i>Table 6.13: Results of benchmarking.</i>	38

Chapter 1 INTRODUCTION

1.1 Software Testing

Software testing is a process of investigate, detect, and found error or bug exist in a software is a process of validation and verification of system (Morgan P. Samaroo, A. Thompson, G. William, 2010). Software testing work in systematically to explore every possible components, systems, or flow for the purpose to found or detect the error exist in the system (Morgan P. Samaroo, A. Thompson, G. William, 2010).

Best practice in software is a much to develop a quality system, because best practice of software testing able to minimal the failure of system and it also help to maintain system marketing value. (Graham, D., Black, R., Van, V.E and Evans,, 2006) Although testing make quality, but software testing is only can found the existing error but not to produce error –free system, due to this a error-free system is almost impossible because exhausting testing, test for every single components in system is impossible. Exhausting testing is impossible is impossible because it may cost a huge among of money and it also take long period to complete.

Software testing is necessary in a system development although software testing may cost a high budget, but a good quality system is needed to reduce system failure, because failure of system may cost higher profit loss than expected, is may cause damage of building, or human life. (Bryce, R.C and Colbourn, C.J., 2006) To avoid the unexpected effect from bad quality system failure, developer is trend to spend for testing process.

Due to the highly cost of testing process, both developer and stake holder are wish to have a most efficient software testing processing, which will cost minimal and found maximum defect.

In history, there are a lot of incidents due to the operation system failure. For example, European Space Agency Ariane 5 exploded incident in June 2006,

(LIONS, 1996) was caused by the software error on floating point overflow this incident was cost up to \$500 million. Another incident was happen in November 2005, the United Kingdom top 10 most wanted criminals in the website was forced to be offline due to exceeding user access. All of the above failures are due the insufficient of software testing; these incidents demonstrate the important of a good software testing.

For the above reason, a new methodology of test case generator has been implemented, t-way testing method. T-way testing is a method to reduce the number of general the test cases by compromising the interaction strength. (Zamli, K. Z., Othman, R. R. and Zabil, M. H. M., 2011)

1.2 Problem Statement

Nowadays, structure of software became complex, huge data included, and multiple functions. The trend of software development is challenging the software tester on software testing process. Increase of size, complex, and function definitely will give challenge to being best practice of software testing. The increase of parameter and data will increase the number of test cases, and indirectly it also increase in cost, as mostly systems are money oriented so this trend should be avoid.

In most of the system, there are multiple data in a function and these functions are combining together to form a work system. In this case, testers are requiring writing test cases follow the possible combination of data exists in the system.

This combination of data might be huge and it may reduce by certain method. One of the exist method to reduce the number of test cases produce was T-way strategic, a strategic that delete redundancy combination data by it specific methodology.

1.3 Objective

This research thesis is aim to develop an alternative flow test data generation for combination testing. Based on this aim, there are three objectives:

1. To develop a prototype to implement the Hill Climbing algorithm for generating of test input data.
2. To investigate the correctness of the Hill Climbing algorithm implementation.
3. To evaluate the performance in term of test size against other existing pairwise testing strategies

1.4 Research Scope

This research thesis will cover the studies of Hill Climbing Algorithm and pairwise testing strategies. The main focus will be on the strategies to reduce test suite size using pairwise testing technique and hill climbing algorithm with achieve of full coverage. NetBeans IDE 7.0.1 was used to develop this prototype.

1.5 Thesis Organization

This thesis consists of 7 chapters, every chapter cover specific issue of the research.

Chapter 1 is Introduction; in chapter 1 will cover the general view about this research. For example, introduction, problem statement, objective, and research scope, are all included in the chapter 1.

Chapter 2 is the Literature Review; in chapter 2 all related issue with the research will be discuss here. In additional, chapter 2 will review on the existing method.

Chapter 3 is Methodology; in chapter 3 the flow of the research will be discuss and Gantt Chart of estimation on the date will be attach too to show a tangible planning of work flow.

Chapter 4 is about design, in design stage the detail design of the thesis prototype will plan and show. The algorithm of how the prototype flow and function will be fully describe in this chapter.

Chapter 5 is Implementation, in this chapter development of prototype will be start and done follow the design on chapter 4. The prototype developed should be functional to enable the next stage of research.

Chapter 6 is Result & Discussion, in this stage a numbers of data will be tested with the developed prototype, and the result will be record and discuss, compare, and verify with the expected result. In begin of this chapter expected result should be define.

Chapter 7, Conclusion will be last and end of research, a final result of the research should be stated and a conclusion with detail explanation of the research should be written.

1.6 Summary

This chapter has discussed the introduction of research on Pairwise Testing Tools Based on Hill Climbing Algorithm. Problem statement, objective, research scope, thesis organization was included in this chapter.

Chapter 2 LITERATURE REVIEW

In Chapter 1, the importance of software testing has been discussed. Building on the material in Chapter 1, this chapter presents the relevant literature review survey. In particular, details of the test case design techniques, existing pairwise testing strategy, covering arrays, and Java script are elaborated in order to justify the current work.

2.1 Theoretical Background

In order to highlight the current work into perspective, this section describes the existing techniques to design the test cases. The techniques that are included in this section are exhausting approach, equivalence partitioning, boundary value analysis, decision table and pairwise testing.

2.1.1 General Exhausting Testing Approach

Exhausting testing was the general method on generating test cases, exhausting approach basically is an approach that will generate all possible test cases for a system. Due to this characteristic, exhausting is always impossible for a huge system, because it is impossible to test all combination. It is costly and time consuming. As example in a restaurant, there is a set menu that included 4 types of foods, each food has several favours.

Table 2.1: Example of exhaustive approach.

<i>Input</i>	<i>Symbolic Representation</i>
<i>Burger</i>	<i>B</i>
<i>Drink</i>	<i>D</i>
<i>Ice-Cream</i>	<i>I</i>
<i>French Fries</i>	<i>F</i>

Table 2.2: Possible input of example.

<i>Symbolic Representation</i>	<i>Possible Input</i>
<i>B</i>	<i>B1, & B2</i>
<i>D</i>	<i>D1, & D2</i>
<i>I</i>	<i>I1, I2, & I3</i>
<i>F</i>	<i>F1, F2, & F3</i>

Table 2.1 and Table 2.2 is the list of the possible input for this system. And the numbers test cases generated should be $2*2*3*3 = 36$. Table 2.3 has shown all the 36 possible test cases, but there is too much test cases included in this case.

Table 2.3: Number of test cases generated by exhaustive approach.

Burger	Drink	Ice-Cream	French fries
B1	D1	I1	F1
B1	D2	I1	F1
B1	D1	I2	F1
B1	D2	I2	F1
B1	D1	I3	F1
B1	D2	I3	F1
B1	D1	I1	F2
B1	D2	I1	F2
B1	D1	I2	F2
B1	D2	I2	F2
B1	D1	I3	F2
B1	D2	I3	F2
B1	D1	I1	F3
B1	D2	I1	F3
B1	D1	I2	F3
B1	D2	I2	F3
B1	D1	I3	F3
B1	D2	I3	F3
B2	D1	I1	F1
B2	D2	I1	F1
B2	D1	I2	F1
B2	D2	I2	F1
B2	D1	I3	F1
B2	D2	I3	F1
B2	D1	I1	F2
B2	D2	I1	F2
B2	D1	I2	F2
B2	D2	I2	F2
B2	D1	I3	F2
B2	D2	I3	F2
B2	D1	I1	F3

B2	D2	I1	F3
B2	D1	I2	F3
B2	D2	I2	F3
B2	D1	I3	F3
B2	D2	I3	F3

2.1.2 Equivalence Partitioning Approach

Equivalence partitioning testing approach is a black box test cases approach, which means only concern on the input and output. Equivalence partitioning will classified into group according to specific characteristic. For each group, only a set of data will be tested due to the similarity of the data. For example:

$$-5 < a < 40$$

$$-12 < b < 35$$

$$0 < c < 20$$

The test cases used will be:

Table 2.4: Test cases implement for Equivalence partitioning approach.

Test	A	B	C
1	-4	-13	-1
2	6	10	7
3	45	50	22

In this approach, the coverage of the flow is very small, so it is not a good testing approach, especially for the cases that contain alternative flow. But this approach has its benefits too, for example it consume least time and cost.

2.2 Survey of Existing Pairwise Strategies

After completed on the theoretical background, the followed section was surveying on existing pairwise strategies.

2.2.1 G2Way

G2Way is one of the tools that apply pairwise testing strategy, G2Way consist of pair generation algorithm and backtracking algorithm (Klaib et al, 2008). Figure 2.1 show the algorithm for pair generation for G2Way strategy.

```

Algorithm Pairs_Generation ( )
1: begin
2:   initialize  $S_p = \{\}$  where  $S_p$  represents the pair set
3:   let  $n_s = \{n_1, \dots, n_m\}$  where  $n_s$  represents the values defined for each parameter,  $m = \text{maximum no of parameters}$ 
4:   let  $p = \{p_1, \dots, p_m\}$  where  $p$  represents the sorted set of sets of values defined for each parameter
5:   for index=0 to  $2^m - 1$ 
6:     begin
7:       let  $b = \text{binary number}$ 
8:        $b = \text{convert index to binary}$ 
9:       if (the no of '1's in  $b = 2$ )
10:        begin
11:          calculate number of possible combinations ( $PC_i$ ) between the partial sets of values
12:          for the shared parameters
13:            begin
14:              multiply  $\{n_s, x, n_s\}$  values from  $n_s$ 
15:              set the bus group (equal to  $PC_i$ ) in the index row to 1
16:            end
17:          end
18:        return  $S_p$ 
19:      end

```

Figure 2.1: Algorithm for pair generation for G2Way (Klaib et al, 2008)

Based on the figure, the loop edge for 2-way interaction will be finding first. Then, the index search will be performed. Considers there are 3 parameters P0, P1, P2 where each parameters have value 2, 3, 1 respectively, the loop search will be 7 where $2^3 - 1$. The index's number will be converting to binary format as in Figure 2.2.

Index	0	1	2	3	4	5	6	7
Binary	000	001	010	011	100	101	110	111

Figure 2.2: Example of index search (Klaib et al., 2008)

The index with two binary one's will be put in the index set. Based on the example in the figure above, index 3, 5, and 6 have two binary ones. Then, each row of

combination of possible pairwise value is generated. There are 3 rows of possible pairwise combination which are P0 and P1, P0 and P2, and P2 and P3.

Row Index	Index		b5	b4	b3	b2	b1	b0
0	3	→	1	1	1	1	1	1
1	5	→	0	0	0	0	1	1
2	6	→	0	0	0	1	1	1

Figure 2.3: Example of row index (Klaib et al., 2008)

Row index 0 is the combination of P0 and P1. The value for P0 is 2 and the value for P1 is 3. Thus the row index store 6 pairs of combination. For row index 1 and 2, 2 pairs stored respectively.

To complete a test suite backtracking algorithm crosses the set of pairwise in iterative ways to combine pairs with usual values of parameter. This algorithm will start to the first define values if pairs cannot be combined existed. The values in pairwise set deleted when the pairs are covered, all pairs are covered only if the pairwise set is empty.

2.2.2 Genetic Algorithm (GA)

Shiba et al. (2004) proposed two new algorithms based on two artificial life techniques to generate test case for combinatorial testing. One of the proposed algorithms is AETG algorithm modification with Genetic Algorithm (GA).

```

Input: A test set.
Output: A test case.
Begin
  Create the initial population  $P$  of candidates.
  Evaluate  $P$ .
  While (stopping condition is not met) {
    Select Elite consisting of  $e$  best individuals from  $P$ .

    Apply Selection to individuals in  $P$  to create  $P_{mating}$ ,
    consisting of  $(n - e)$  individuals.

    Crossover  $P_{mating}$ .
    Mutate  $P_{mating}$ .

    Copy the all individuals of  $P_{mating}$  to  $P$ ,
    replacing the worst  $(n - e)$  individuals in  $P$ .

    Evaluate  $P$ .
    If (stagnation condition is met) Mutate  $P$  massively.
  }
  Return the best test case found.
End.

```

Figure 2.4: :Outline of the proposed GA (Shiba et al., 1997)

GA impersonates the growth of single celled organism. Generally there are four important aspects in GA that are chromosome encoding and fitness function, selection, crossover and mutation, and GA treated a test case as a chromosome, and the fitness function is used to approximate the goodness of the candidate solution. Fitness function $F(S)$ for a test case S is defined as the number of t -way combinations that covered by S but not covered by the given test set. Based on the algorithm shows in the figure, the initial population of test cases' candidate, P is created randomly.

The population is then evaluated. The best test case from the population is selected using elite strategy. To the remaining test case, *Selection* is applied to create

population mating. After that, the test case in mating pool is being crossover and the value of a position is replaced with another value by mutation randomly. After crossover and mutation, the test cases in mating pool are copy to the P to replace the worst test case. Then, the P is evaluated again. P is mutated vastly if the stagnation condition is met. In this case, the stagnation condition is the generations' number that passes ever since the last improvement in solution. These actions are performed continuously until the stopping condition met.

2.3 Summary

This chapter has review on the existing testing theoretical background and general idea of pairwise testing, and existing pairwise testing strategy.

Chapter 3 METHODOLOGY

In chapter 2, the review on the existing testing strategies has been discussed. In this chapter, the methodology of PTCA has been discussed in detail.

3.1 Methodology

Research methodology is an explanation of the process included to complete the research project. There are four stages included in this research project, for example, Literature Review, Analysis and Design, Development, and Analysis the result. As shown in figure3.1.

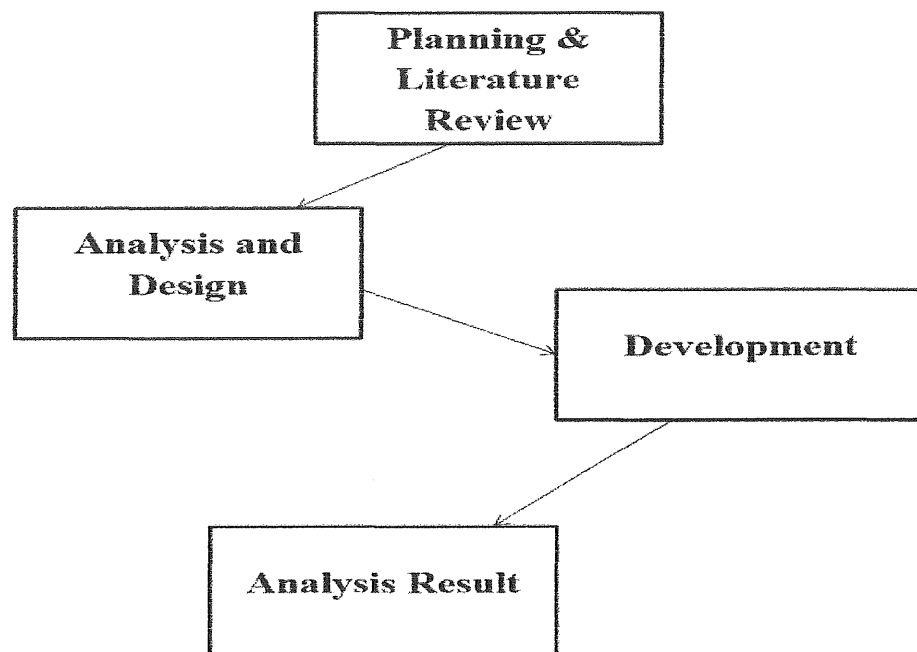


Figure 3.1: Overview flow of methodology

The first stage, Literature Review is a stage where will study of the existing problem on the past testing process faced. The concern on this stage is about the t-way testing strategies, and how this strategies help in reduce the number of test cases.

Secondly, we will proceed into Analysis and Design process. In here we are going to analyst the possible strategies to improve the testing process. In addition here also will include the design of the new possible strategies to reduce the number of test cases.

Next process is the development stage; this stage is mainly focused on to implement the possible strategies. Furthermore, in this stage test cases generator will be develop base on the new strategies.

The final stage was the Analysis Result process, in here collecting of the result producing by the tool (test case generator) and the analysis of these result will be proceed. Moreover, the result of this research will be answer in this stage.

3.2 Hardware & Software

Table 3.1: List of software used

Software Used	Description
Net Bean	Use in development the prototype in Java.
Microsoft Office	Use in documentation, e.g writing report, drawing chart and e.t.c

Table 3.2 : List of Hardware used

Hardware Used	Description
Laptop	Use in development the prototype in Java. Use in writing report.
Stationary	Use to record minute of meeting. Use on drafting.

Table 3.1 & table 3.2 has listed the software and hardware used for PTCA. PTCA was developed using java programing language with NetBean as platform. During the research, MS office was used as report writing, result recording.

For hardware, an Intel i5-2410M, 2.30GHz with 8GB ram laptop was used to develop the prototype and report writing.

3.3 Summary

This chapter discussed on the flow of the PTCA research, besides hardware and software used has been included too.

Chapter 4 DESIGN

In chapter3, detail of the tools and research flow has been discussed. In this chapter, detail design of PTCA has been discussed.

3.1 Design Consideration

The system designed is selected the loop from first test case in the test cases, and get the coverage of each test case, then save the most coverage test case as the best test case.

3.2 Development of PTCA Strategy

PTCA is tools that function based on pairwise testing strategy and hill climbing algorithm. Figure4.1 show the overall design of PTCA.

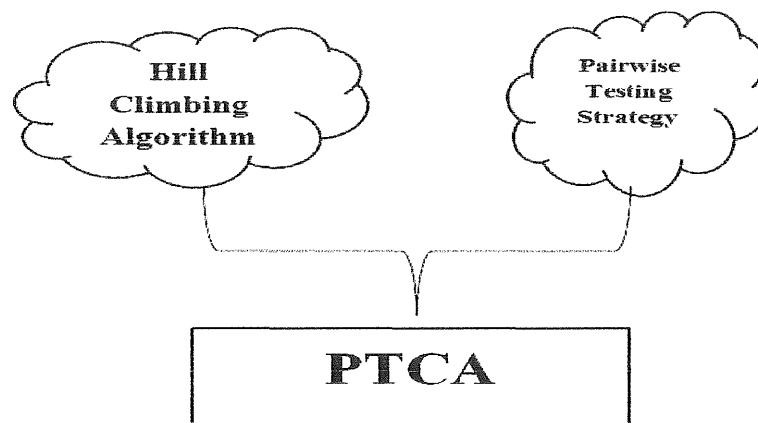


Figure 4.1: Overall design of PTCA

PTCA strategy design used pairwise testing strategy to reduce the test case generated, based on the hill climbing algorithm to as process method, to achieve full coverage of the combination.

3.3 Hill Climbing Algorithm

```

1. for (int i=0; i<sample.length; i++)
2.   for (int r=1; r<sample.length; r++)
3.     compare sample[i] with sample[r]
4.     if sample[i]< sample[r]
5.       temp= sample[i]
6.       sample[i]=sample[r]
7.       sample[r]=temp
8.   end

```

Figure 4.2 : Algorithm of hill climbing algorithm

Hill climbing algorithm is an incremental concept algorithm which would start from the basic to the peak. On PTCA, hill climbing algorithm was used on merging process. Which PTCA will retrieve the 1st combination from the combination list to merge with the data in the pair list.

3.4 Weight Counting Method

Weight counting is a identified in PTCA, for each loop of merging process the weight will recorded and compare with the previous weight. The heaviest weight combination consider as the best.

3.5 Pairwise Testing Strategy

```

1. Begin
2. let pair as ArrayList[]
3. let pair={-1,-1,-1-1,...}
4. let int time =0
5. for (int s1=0; s1< pair.length; s1++){
6.   for (int s2=1; s2<=pair.length; s2++){
7.     for(int value1=0;value1< parameter_1;value_1++){
8.       for(int value1=0;value2< parameter_2;value_2++){
9.         replace pair.[time].[s1]= value_1
10.        replace pair.[time].[s2]= value_2
11.        time++
12.      }}}}
13. End.

```

Figure 4.3: Algorithm of pairwise strategy

Pairwise strategy is a strategy that generating a list of combination with only two known value. Figure4.3 show the algorithm set the -1 values as unknown value, and substitute the parameter value in the list to generate the pair list.

3.6 Development Strategy

In the chapter the development is based on hill climbing algorithm. Generally, this system is searching for the best test cases following the hill climbing algorithm, where the algorithm will pick the first sample and find the coverage of that sample. Then the coverage is then compare with the others coverage

```

1. Input Parameter
2. Let pair as pairList[]
3. Let TC as TestCases[]
4. while TC not empty
5. for (y=0;y<Pair.size();y++)
6. let weight=0 as integer.
7. for (x=0;x<TC.size();x++)
8. Compare Pair[y]with TC[x]
9. if Pair[y] similiar to TC[x]
10. weight +=1
11. if weight> bestweight[x]
12. let Best as bestList[]
13. let c as integer, c=0
14. Best[c] = TC[x]
15. c++
16. Remove TC[x]|
17. output Best

```

Figure 4.4: Algorithm of the system

3.7 Summary

This chapter is described the design of this system, it including the algorithm and description of the flow of the system.

Chapter 5 IMPLEMENTATION

In previous chapter, the detail design of this system has been discussed. So, on this chapter detail about implementation of this system will be discuss. First of all, this system is develop using NetBeans IDE 7.0.1 on an Intel core i5-2410M, 2.3GHz CPU with 8GB of RAM Notebook. This system is develop using Java programing language.

5.1 Implementation of Exhaustive Testing

The hill climbing algorithm is active after the input is specified. The study case applied here was 4 parameter with 3 values each, the input is specified as 3,3,3,3 (refer Figure 5.1).

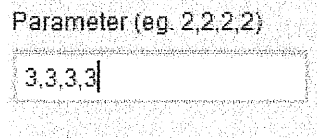


Figure 5.1: 4 inputs with 3 valued

Then the input will process to generate a possible combination of these 4 parameters.(refer figure 5.2,5.3,5.4)

```

Case 0 = [0, 0, 0, 0]
Case 1 = [0, 0, 0, 1]
Case 2 = [0, 0, 0, 2]
Case 3 = [0, 0, 1, 0]
Case 4 = [0, 0, 1, 1]
Case 5 = [0, 0, 1, 2]
Case 6 = [0, 0, 2, 0]
Case 7 = [0, 0, 2, 1]
Case 8 = [0, 0, 2, 2]
Case 9 = [0, 1, 0, 0]
Case 10 = [0, 1, 0, 1]
Case 11 = [0, 1, 0, 2]
Case 12 = [0, 1, 1, 0]
Case 13 = [0, 1, 1, 1]
Case 14 = [0, 1, 1, 2]
Case 15 = [0, 1, 2, 0]
Case 16 = [0, 1, 2, 1]
Case 17 = [0, 1, 2, 2]
Case 18 = [0, 2, 0, 0]
Case 19 = [0, 2, 0, 1]
Case 20 = [0, 2, 0, 2]

```

Figure 5.2: The possible combination of the input (1)


```

Case 21 = [0, 2, 1, 0]
Case 22 = [0, 2, 1, 1]
Case 23 = [0, 2, 1, 2]
Case 24 = [0, 2, 2, 0]
Case 25 = [0, 2, 2, 1]
Case 26 = [0, 2, 2, 2]
Case 27 = [1, 0, 0, 0]
Case 28 = [1, 0, 0, 1]
Case 29 = [1, 0, 0, 2]
Case 30 = [1, 0, 1, 0]
Case 31 = [1, 0, 1, 1]
Case 32 = [1, 0, 1, 2]
Case 33 = [1, 0, 2, 0]
Case 34 = [1, 0, 2, 1]
Case 35 = [1, 0, 2, 2]
Case 36 = [1, 1, 0, 0]
Case 37 = [1, 1, 0, 1]
Case 38 = [1, 1, 0, 2]
Case 39 = [1, 1, 1, 0]
Case 40 = [1, 1, 1, 1]
Case 41 = [1, 1, 1, 2]
Case 42 = [1, 1, 2, 0]
Case 43 = [1, 1, 2, 1]
Case 44 = [1, 1, 2, 2]
Case 45 = [1, 2, 0, 0]
Case 46 = [1, 2, 0, 1]
Case 47 = [1, 2, 0, 2]
Case 48 = [1, 2, 1, 0]
Case 49 = [1, 2, 1, 1]
Case 50 = [1, 2, 1, 2]

```

Figure 5.3 : The possible combination of the input (2)

```

Case 51 = [1, 2, 2, 0]
Case 52 = [1, 2, 2, 1]
Case 53 = [1, 2, 2, 2]
Case 54 = [2, 0, 0, 0]
Case 55 = [2, 0, 0, 1]
Case 56 = [2, 0, 0, 2]
Case 57 = [2, 0, 1, 0]
Case 58 = [2, 0, 1, 1]
Case 59 = [2, 0, 1, 2]
Case 60 = [2, 0, 2, 0]
Case 61 = [2, 0, 2, 1]
Case 62 = [2, 0, 2, 2]
Case 63 = [2, 1, 0, 0]
Case 64 = [2, 1, 0, 1]
Case 65 = [2, 1, 0, 2]
Case 66 = [2, 1, 1, 0]
Case 67 = [2, 1, 1, 1]
Case 68 = [2, 1, 1, 2]
Case 69 = [2, 1, 2, 0]
Case 70 = [2, 1, 2, 1]
Case 71 = [2, 1, 2, 2]
Case 72 = [2, 2, 0, 0]
Case 73 = [2, 2, 0, 1]
Case 74 = [2, 2, 0, 2]
Case 75 = [2, 2, 1, 0]
Case 76 = [2, 2, 1, 1]
Case 77 = [2, 2, 1, 2]
Case 78 = [2, 2, 2, 0]
Case 79 = [2, 2, 2, 1]
Case 80 = [2, 2, 2, 2]

```

Figure 5.4: The possible combination of the input (3)

Figure 5.2, figure 5.3, and figure 5.4 show the exhaustive combination of the 4 parameter with 3 values. Form these figures, the number of exhaustive combination is 81 (form 0 to 80).

5.2 Implementation of Pairwise Test Strategy

Using Pairwise Strategy, the system will generate all possible pairwise combination as figure.

```

***** Pair List *****
PairList 0 = [0, 0, -1, -1]
PairList 1 = [0, 1, -1, -1]
PairList 2 = [0, 2, -1, -1]
PairList 3 = [1, 0, -1, -1]
PairList 4 = [1, 1, -1, -1]
PairList 5 = [1, 2, -1, -1]
PairList 6 = [2, 0, -1, -1]
PairList 7 = [2, 1, -1, -1]
PairList 8 = [2, 2, -1, -1]
PairList 9 = [0, -1, 0, -1]
PairList 10 = [0, -1, 1, -1]
PairList 11 = [0, -1, 2, -1]
PairList 12 = [1, -1, 0, -1]
PairList 13 = [1, -1, 1, -1]
PairList 14 = [1, -1, 2, -1]
PairList 15 = [2, -1, 0, -1]
PairList 16 = [2, -1, 1, -1]
PairList 17 = [2, -1, 2, -1]
PairList 18 = [0, -1, -1, 0]
PairList 19 = [0, -1, -1, 1]
PairList 20 = [0, -1, -1, 2]
PairList 21 = [1, -1, -1, 0]
PairList 22 = [1, -1, -1, 1]
PairList 23 = [1, -1, -1, 2]
PairList 24 = [2, -1, -1, 0]
PairList 25 = [2, -1, -1, 1]
PairList 26 = [2, -1, -1, 2]
PairList 27 = [-1, 0, 0, -1]
PairList 28 = [-1, 0, 1, -1]

```

Figure 5.5: The list when the input converting into pairwise (1)

```

PairList 29 = [-1, 0, 2, -1]
PairList 30 = [-1, 1, 0, -1]
PairList 31 = [-1, 1, 1, -1]
PairList 32 = [-1, 1, 2, -1]
PairList 33 = [-1, 2, 0, -1]
PairList 34 = [-1, 2, 1, -1]
PairList 35 = [-1, 2, 2, -1]
PairList 36 = [-1, 0, -1, 0]
PairList 37 = [-1, 0, -1, 1]
PairList 38 = [-1, 0, -1, 2]
PairList 39 = [-1, 1, -1, 0]
PairList 40 = [-1, 1, -1, 1]
PairList 41 = [-1, 1, -1, 2]
PairList 42 = [-1, 2, -1, 0]
PairList 43 = [-1, 2, -1, 1]
PairList 44 = [-1, 2, -1, 2]
PairList 45 = [-1, -1, 0, 0]
PairList 46 = [-1, -1, 0, 1]
PairList 47 = [-1, -1, 0, 2]
PairList 48 = [-1, -1, 1, 0]
PairList 49 = [-1, -1, 1, 1]
PairList 50 = [-1, -1, 1, 2]
PairList 51 = [-1, -1, 2, 0]
PairList 52 = [-1, -1, 2, 1]
PairList 53 = [-1, -1, 2, 2]

```

Figure 5.6: The list when the input converting into pairwise (2)

Figure 5.5, & Figure 5.6 show the list of pairwise combination generate by pairwise testing strategy. In the combination, -1 represent the unknown with no value. For each combination only two value is defined.

Chapter 6 RESULT AND DISCUSSION

After implementation, PTCA strategy will be evaluated to verify the effectiveness and performance in order to achieve and match the goals and objective. Initially, few tests are done to test the correctness of the PTCA function, testing the algorithm has demonstrated correctly, besides a test on the reducing test cases generated is done to prove that PTCA is able to reduce the number of test cases.

6.1 Demonstration of Correctness

Objective of PTCA is to investigate the correctness of the implementation of the hill climbing algorithm. In order to verify that PTCA achieve this goal, two experiments as conducted.

A web-based configuration example is selected as a case study for the first experiment. This selection is base in the fact that the input is used by other strategies such as G2Way (Klaibt et al., 2008). The web-based configuration example has 4 parameter and each parameters have 3 values (see Table 6.1).

Table 6.1: 4 input with 3 valued

P1	P2	P3	P4
Netscape	Windows	LAN	Local
IE	Macintosh	PPP	Networked
Firefox	Linux	ISDN	Screen

As PTCA will only generate test cases in number form, so the following input is convert into number.

Table 6.2 : Symbol of each input

P1	P2	P3	P4
Netscape =0	Windows=0	LAN=0	Local=0
IE =1	Macintosh =1	PPP=1	Networked=1
Firefox=3	Linux=2	ISDN=2	Screen=2

The test suite as shown in figure 6.1 was screen capture from PTCA screen, which been generated based on the web-based configuration example in Table 6.1 produces 12 test cases.

```

run
3, 3, 3, 3

***** Result *****
0 Best= [0, 0, 0, 0]
1 Best= [0, 1, 1, 1]
2 Best= [0, 2, 2, 2]
3 Best= [1, 0, 1, 2]
4 Best= [1, 1, 2, 0]
5 Best= [1, 2, 0, 1]
6 Best= [2, 0, 2, 1]
7 Best= [2, 1, 0, 2]
8 Best= [2, 2, 1, 0]
9 Best= [0, 1, 0, 1]
10 Best= [0, 2, 0, 0]
11 Best= [2, 0, 1, 0]
Uncover= 0
BUILD SUCCESSFUL (total time: 5 seconds)

```

Figure 6.1: result screen on PTCA

Table 6.3 was the result for the PTCA.:

Table 6.3: Result

T#	P1	P2	P3	P4
1	Netscape	Windows	LAN	Local
2	Netscape	Macintosh	PPP	Networked
3	Netscape	Linux	ISDN	Screen
4	IE	Windows	PPP	Screen
5	IE	Macintosh	ISDN	Local
6	IE	Linux	LAN	Networked
7	Firefox	Windows	ISDN	Networked
8	Firefox	Macintosh	LAN	Screen
9	Firefox	Linux	PPP	Local
10	Netscape	Macintosh	LAN	Networked
11	Netscape	Linux	LAN	Local
12	Firefox	Windows	PPP	Local

Based on table 6.3 and figure 6.1, there are six possible interactions for the parameters. Those interactions are between P1 and P2, P1 and P3, P1 and P4, P2 and P3, P2 and P4, and P3 and P5. The expected total interaction pairs should be 54 based on these interactions.

In order to investigate the correctness of PTCA algorithm, the test suite generated by PTCA is analysed. The correctness of PTCA algorithm is proved when all interaction pairs are covered at least once

Table 6.4: List of total interaction pairs

Pair Combination	T#	Pair Combination	T#
Netscape, Windows	1	Windows, LAN	1
Netscape, Macintosh	2,10	Windows, PPP	4,12
Netscape, Linux	3,11	Windows, ISDN	7

Netscape, LAN	1,10,11	Windows, Local	1,12
Netscape, PPP	2	Windows, Networked	7
Netscape, ISDN	3	Windows, Screen	4
Netscape, Local	1,11	Macintosh, LAN	8,10
Netscape, Networked	2,10	Macintosh, PPP	2
Netscape, Screen	3	Macintosh, ISDN	5
IE, Windows	4	Macintosh, Local	5
IE, Macintosh	5	Macintosh, Networked	2,10
IE, Linux	6	Macintosh, Screen	8
IE, LAN	6	Linux, LAN	6,11
IE, PPP	4	Linux, PPP	9
IE, ISDN	5	Linux, ISDN	3
IE, Local	5	Linux, Local	9,11
IE, Networked	6	Linux, Networked	6
IE, Screen	4	Linux, Screen	3
Firefox, Windows	7,12	LAN, Local	1,11
Firefox, Macintosh	8	LAN, Networked	6,10
Firefox, Linux	9	LAN, Screen	8

Table 6.4 shows the list of total interaction pairs with the test case generated by PTCA that covered the pairs. Referring to the table above, it can be seen that the interaction pairs are covered at least once. This means that the generated test cases include all pairs. Since none pairs are missing, it means PTCA strategy algorithms are correct.

The second experiment used an online pizza ordering system example as a case study. This example has an input consists of 3 parameter with each value of 2 as shown in Table 6.5

Table 6.5: 3 parameter with each value of 2

Crust	Flavours	Toppings
Classic Hand Tossed	Vegetarian	Pineapples
Crunchy Thin	Pepperoni	Beef

Referring to Table 6.6, the test suite was generated based on the input specification stated in the Table 6.5. PTCA generated the test suite that consists of 6 test cases for this experiment.

```

2,2,2

***** Result *****
0 Best= [0, 0, 0]
1 Best= [0, 1, 1]
2 Best= [1, 0, 1]
3 Best= [1, 1, 0]
4 Best= [0, 0, 1]
5 Best= [1, 0, 0]
Uncover= 0
BUILD SUCCESSFUL (total time: 2 seconds)
|

```

Figure 6.2: result screen in PTCA

Table 6.6: Result

T#	Crust	Flavours	Toppings
1	Classic Hand Tossed	Vegetarian	Pineapples
2	Classic Hand Tossed	Pepperoni	Beef
3	Crunchy Thin	Vegetarian	Beef
4	Crunchy Thin	Pepperoni	Pineapples
5	Classic Hand Tossed	Vegetarian	Beef
6	Crunchy Thin	Vegetarian	Pineapples

Based on this case study, there are 3 possible interactions that are Crust and Flavours, Crust and Toppings, and Flavours and Topping. Thus, the expected interaction pairs are 12.

The aim for this experiment is similar with the first experiment which is to prove that the PTCA's algorithms are correct. Table 6.6 lists the interaction pairs for the case study with the test cases that covered them. Since, the generated test suite covered all pairs at least once and no missing pairs, the algorithms for PTCA strategy are correct.

Table 6.7 : List of total interaction pairs

Pair Combination	T#
Classic Hand Tossed, Vegetarian	1,5
Classic Hand Tossed, Pepperoni	2
Classic Hand Tossed, Pineapples	1
Classic Hand Tossed, Beef	2,5
Crunchy Thin, Vegetarian	3,6
Crunchy Thin, Pepperoni	4
Crunchy Thin, Pineapples	4
Crunchy Thin, Beef	3
Vegetarian, Pineapples	1, 6
Vegetarian, Beef	3,5
Pepperoni, Pineapples	4
Pepperoni, Beef	2

6.2 Demonstration Of Reducing Test Suite Size

Pairwise testing objective is to reduce the size of the test suite. In order to prove PTCA can reduce the test suite size, an experiment was conducted. This experiment is done in the following condition:

- The results are obtain by using Windows 7 Home Premium with Intel Core i5-2410M, 2.3GHz CPU and 8GB Ram with NetBeans IDE 7.0.1 installed.

Table 6.8: Input specification for test A, B, C, D, and F

Tests	Input Specification
A	4 2-valued parameters
B	2 2-valued parameters, 3 3-valued parameters
C	2 5-valued parameters, 3 6-valued parameters
D	2 3-valued parameters, 2 5-valued parameters, 3 6-valued parameters
F	3 3-valued parameters, 3 5-valued parameters, 3 6-valued parameters

The same inputs then run independently 5 times and every result was recorded in and compared with the test suite cases sizes of exhaustive testing.

Table 6.9: Results of test suite size generated with number of runs

Test\ num of run	1	2	3	4	5
A	7	7	7	7	7
B	13	13	13	13	13
C	49	49	49	49	49
D	44	44	44	44	44
F	48	48	48	48	48

The table above shows the results of the test suite size against the number of run for the 5 tests. It is noticeable that number of test cases is constant, it mean that PTCA would not effected by the number of run, it will generated a constant result.

For the Test A, the test suite sizes that are generated exhaustively are 16 (2x2x2x2). Using PTCA the test suite size is reduced from 16 to 7. While PTCA generate test suite size of 13 for Test B that is smaller than exhaustive testing test suite size that are 108. PTCA generates test suite size of 49 for Test C and test suite size of 5400 for Test D. The differences between test suite size of exhaustive testing and test suite size generated by PTCA are shown in Table 6.9.

Table 6.10: Differences between exhaustive testing test suite size and PTT test suite size

Tests	Test Suite Sizes		Total reduce	Percentage of reduce (%)
	Exhaustive	PTCA		
A	16	7	9	56.25
B	108	13	95	87.96
C	5400	49	5351	99.09
D	48600	44	48556	99.9
F	729000	48	7289952	99.9

Based on table above, PTCA can reduce the number test up to 99% compare to exhaustive test strategy.

6.3 Results of PTCA Benchmarking

Evaluating the performance of PTCA in term of test size against other existing pairwise testing strategies is one of the objectives of PTCA. The benchmarking is done in the following condition:

- The results are obtain by using Windows 7 Home Premium with Intel Core i5-2410M, 2.3GHz CPU and 8GB Ram with NetBeans IDE 7.0.1 installed.
- In order to benchmark PTCA strategy in term of the test size generated, the same data that is used to benchmark G2Way strategy is used. G2Way strategy used eight sets of data to compare the strategy with other strategies that are AETG, AETGm, IPO, SA, GA, ACA, PTT, and ALL Pairs (Klaib et al., 2008). The sets of data are as follow:

Table 6.11: Input specification for 8 tests

Tests	Input Specification
S1	3 3-valued parameters
S2	4 3-valued parameters
S3	13 3-valued parameters
S4	10 10-valued parameters
S5	10 15-valued parameters

S6	20 10-valued parameters
S7	10 5-valued parameters
S8	1 5-valued parameters, 8 3-valued parameters, and 2 2-valued parameters

Table 6.12: Result of the comparison data in term of the test size (Klaib et al., 2008)

Sys	AET	AETGm	IPO	SA	GA	ACA	All-Pairs	G2-Way	PPT	PTCA
S1	NA	NA	NA	NA	NA	NA	10	10	18	11
S2	9	11	9	9	9	9	10	10	37	11
S3	15	17	17	16	17	17	22	19	383	23
S4	NA	NA	169	NA	157	159	177	160	4209	NA
S5	NA	NA	361	NA	NA	NA	390	343	9817	NA
S6	180	198	212	183	227	225	230	200	16969	NA
S7	NA	NA	47	NA	NA	NA	49	46	899	46
S8	19	20	NA	15	15	16	21	23	268	22

PTCA has a limitation on the memory; it can't effort for the large data such as S4, S5, S6, and S7 which has a large number of data. For those test data PTCA is out of function.

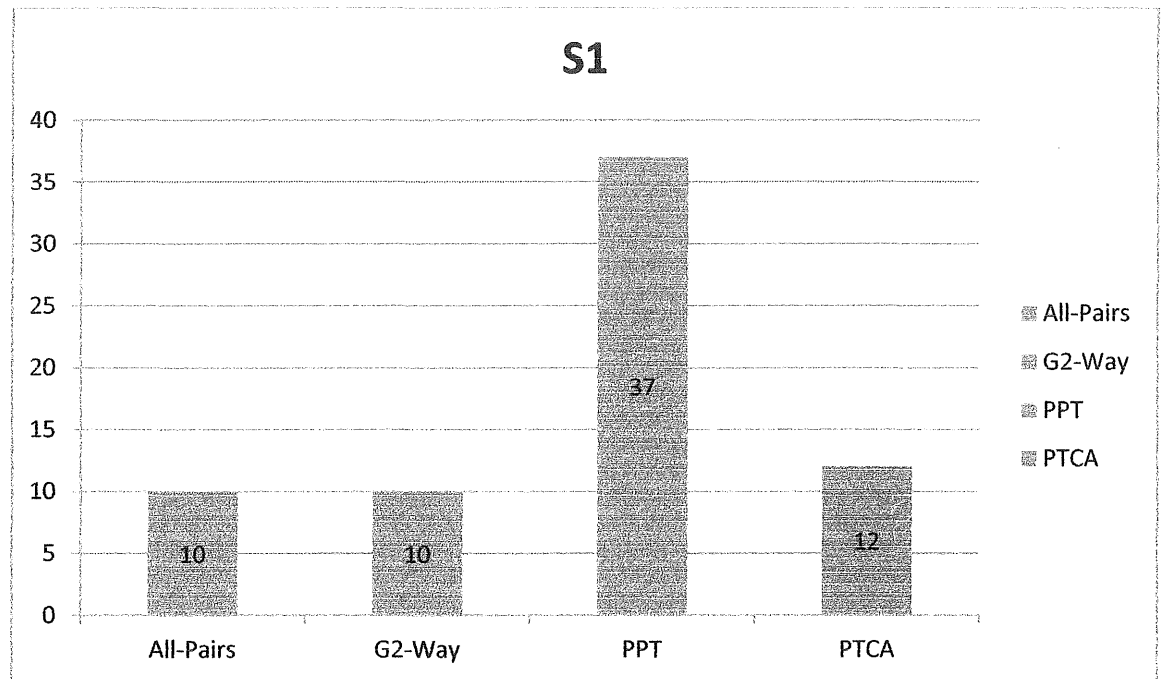


Figure 6.3: Result of S1

For S1 (3 inputs with 3 value), PTCA generate 12 test cases, will is better performance than PPT which generated 37 test cases with the same input, but PTCA was a bit low efficient compare to All-pair and G2-Way which successful generate only 10 test cases for same input.

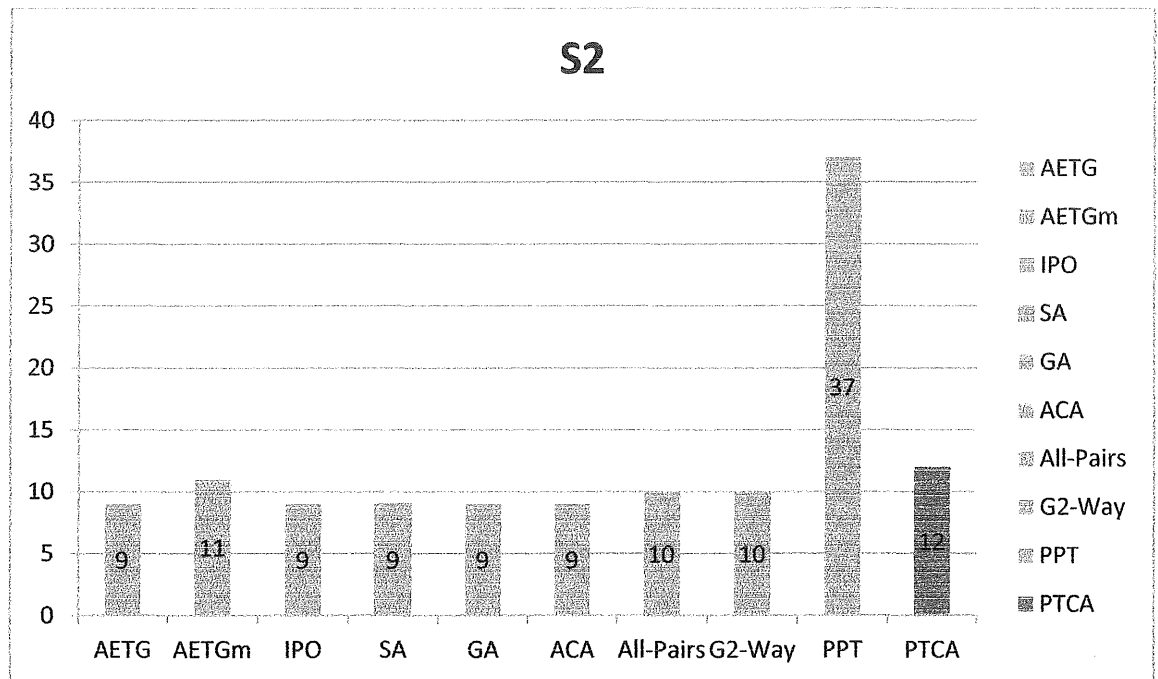


Figure 6.4: Result of S2

For S2 (4 inputs with 3 valued parameters), PTCA generated 12 test cases which is much better performance compare to PTT which generated 37 test cases for same input, besides PTCA result was closed to other test strategies which generate around range of 9 test cases to 12 test cases.

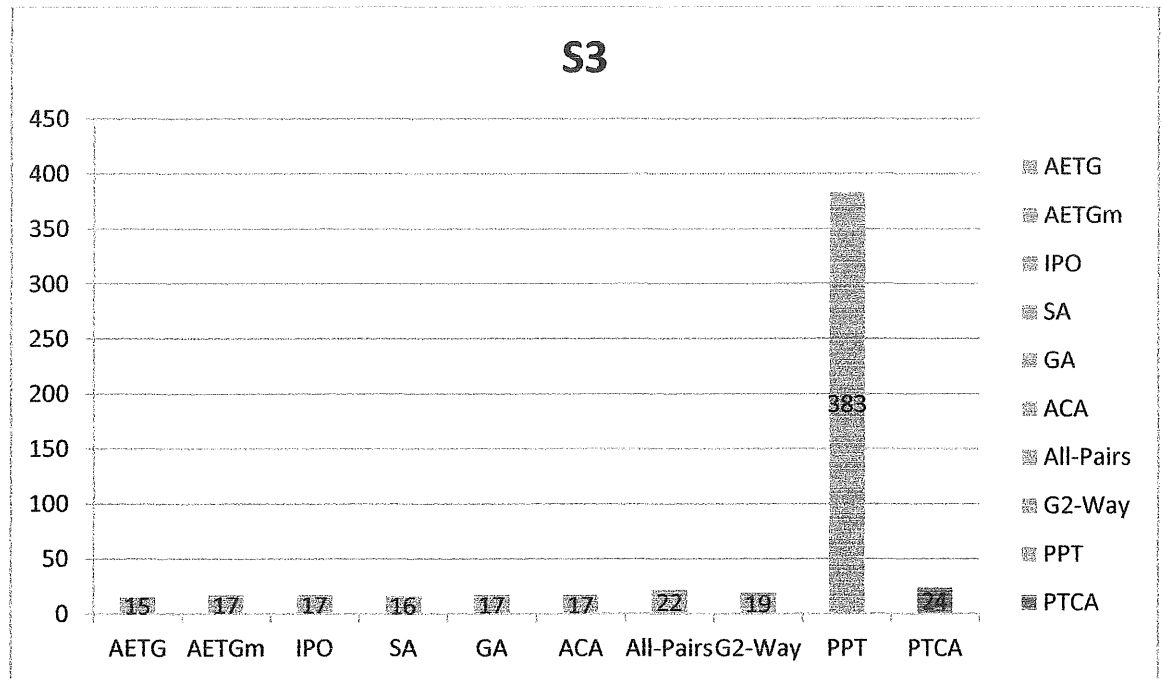


Figure 6.5: Result of S4

For S3 (13 inputs with 3 valued), PTCA generate 24 test cases which slightly high compare to AETG with 15 test cases, AETGm, IPO, GA, ACA, both 17 test cases, SA with 16 test cases, All-Pair with 22 test cases, and G2Way with 19 test cases. In other hand, PTCA is generated much better result compare to PTT which generate 383 test cases for the same input.

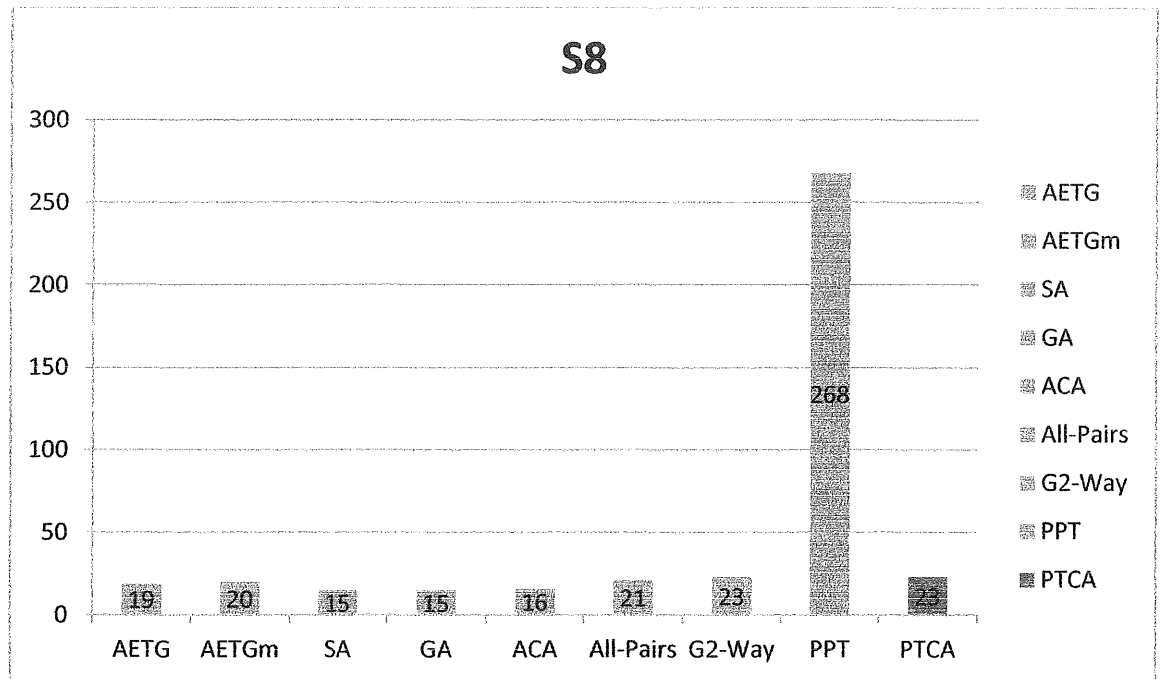


Figure 6.6: Result of S8

For S8 (1 input of 5-valued parameters, 8 inputs of 3-valued parameters, and 2 inputs of 2-valued parameters), PTCA was class as good performance strategy which generate 23 test cases, a closed result with the others test strategies except PTT, PTCA beat PTT as PTT generate a high numbers of test cases with 268.

Table 6.13: Results of benchmarking

Sys	AETG	AETGm	IPO	SA	GA	ACA	All-Pairs	G2-Way	PPT	PTCA
S1	NA	NA	NA	NA	NA	NA	10	10	18	12
S2	9	11	9	9	9	9	10	10	37	12
S3	15	17	17	16	17	17	22	19	383	24
S4	NA	NA	169	NA	157	159	177	160	4209	NA
S5	NA	NA	361	NA	NA	NA	390	343	9817	NA
S6	180	198	212	183	227	225	230	200	16969	NA
S7	NA	NA	47	NA	NA	NA	49	46	899	47
S8	19	20	NA	15	15	16	21	23	268	23

Table 6.13 depicts the result of PTCA (shaded column) with others strategies, as conclusion for PTCA in performance, PTCA can be classed as the best performance strategy which PTCA generate a closed result with the best performance strategy result. But PTCA was face limitation in data size. There are few character of PTCA:

- PTCA is performance closely to the best test strategy, which will always generate result near to the best result.
- PTCA is a fix test strategy, it will generate the constant result for the same input compare to other strategies which may need to

run several times to get the best result, PTCA result is constant, therefore user can run one and record the result, this character save the user effort and time to retrieve result.

- PTCA has a limitation for huge data process, for huge data such as S4 (10 input with 10 valued parameters) and above PTCA would not able to process this huge data due to the lack for memory.

6.4 Summary

This chapter discussed on the real time performance of PTCA and comparison of PTCA with others test strategies was done. Based on the result gather from the test, PTCA produce a good result for most of the test data, but there is some exception too, such as PTCA fail to process with few huge test data due to the lacking in memory.

Chapter 7 CONCLUSION

In the earlier chapter, a detail explanation on the flow in working on completing this thesis had been discussed including reviewing, developing PTCA strategy, experimental results and discussions. In this chapter, the research work on PTCA strategy is concluded with future work.

7.1 Conclusion

The objective of this thesis is to develop PTCA strategy that able to reduce test cases generated based on pairwise test strategy and hill climbing algorithm. To achieve this goal a list of objective has been set in the early of the thesis as below:

1. Develop a prototype to implement the Hill Climbing algorithm for generating of test input data.
2. To investigate the correctness of the Hill Climbing algorithm implementation.
3. To evaluate the performance in term of test size against other existing pairwise testing strategies

As conclusion, PTCA has achieved the entire above objectives. When compare with existing strategy PTCA is generated a closed result, so PTCA performance is good enough just there are some failure for PTCA in process with huge data. Thus improvement is needed for PTCA to solve this problem.

7.2 Future Work

For the further enhancement, there are few research area that could be considered:

1. Improve PTCA process time

Although PTCA was produce a very closed result with the current best result but PTCA has a weakness of process time, PTCA was taking long period of process time to generate the result, thus research on reduce the complexity of PTCA is needed.

2. Memory lacking

PTCA was facing problem when execute with the huge data set, a few research may needed to solve this problem such as reduce the data keep in the process and enhancement of the memory allocated in PTCA.

REFERENCES

- Bach, J. AllPairs test case generation tool (online). <http://www.satisfice.com/tools.shtml> (17 November 2013)
- Bryce, R.C. and Colbourn, C.J. 2006. Prioritized interaction testing for pair-wise coverage with seeding and constraints. *Information and Software Technology*, **48**: 960-970.
- Cohen, D.M., Dalal, S.R., Fredman, M.L. and Patton, G.C. 1997. The AETG system: an approach to testing based on combinatorial design. *Software Engineering, IEEE Transactions on Software Engineering*, **23**(7): 437-444.
- Deitel, P.J. and Deitel, H.M. 2009. *Internet & World Wide Web How To Program*. Fourth Edition. New Jersey: Pearson Education , Inc
- Goodman, D., Morrison, M. and Eich B. 2007. *JavaScript Bible*. Sixth Edition. Wiley.
- Graham, D., Black, R., Van, V.E. and Evans, I. 2006. *Foundations of Software Testing: ISTQB Certification*. Cengage Learning.
- Grindal, M, Offutt, J. and Andler, S.F. 2005. Combination testing strategies: a survey. *Journal of Software Testing, Verification, and Reliability*. **15**: 167-199.
- Jun, Y. and Jian, Z. 2006. Backtracking algorithms and search heuristics to generate test suites for combinatorial testing. *Computer Software and Applications Conference, 2006.*, 1:385-394.
- Klaib, M.F.J., Zamli, K.Z., Isa, N.A.M., Younis, M.I. and Abdullah, R. 2008. G2Way a backtracking strategy for pairwise test data generation. *Software Engineering Conference, 2008*, pp. 463-470.
- Kodgirwar, A.B. and Raval, P.N. 2013. Survey of techniques used for combinatorial testing as well as pair wise testing. *ASM's International E-Journal of Ongoing Research in Management And IT*, INCON13-IT-031: 1-12.
- Morgan, P., Samaroo, A., Thompson, G. and Williams, P. 2010. *Software Testing: An ISTQB-ISEB Foundation Guide*. Hambling, B. Second Edition. UK: British Informatics Society Limited (BISL).
- Rabbi, K.F., Khatun, S., Yaakub, C.Y., and Klaib, M.F.J. 2011. EPS2Way: an efficient pairwise test data generation strategy. *International Journal on New Computer Architectures and Their Application (IJNCAA)*. **1**(4): 1099-1109.
- Shiba, T., Tsuchiya, T. and Kikuno, T. 2004. Using artificial life techniques to generate test cases for combinatorial testing. *Computer Software and Applications Conference, 2004*, **71**: 72-77.
- Pat Langley, John H. Gennari. Hill-Climbing Theories of Learning. Irvine Computational Intelligence Project, Department of Info & Computer Science, University of California, Irvine.

Younis, M.I., Zamli, K.Z. and Isa, N.A.M. 2008a. Algebraic strategy to generate pairwise test set for prime number parameters and variables. *Information Technology, ITSIm 2008*, pp. 1-4.

Younis, M.I., Zamli, K.Z. and Isa, N.A.M. 2008b. Generating pairwise combinatorial test set using artificial parameters and values. *Information Technology, ITSIm 2008*, pp. 1-8.

Younis, M.I., Zamli, K.Z., Klaib, M.F.J., Soh, Z.H.C., Abdullah, S.A.C. and Isa, N.A.M. 2010. Assessing IRPS as an efficient pairwise test data generation strategy. *International Journal of Advanced Intelligence Paradigms*, **2**(1): 90-104.

Yu, L. and Tai, K.C. 1998. In-parameter-order: a test generation strategy for pairwise testing. *High-Assurance Systems Engineering Symposium, 1998*, pp. 254-261.

Zamli, K.Z., Younis, M. I., Ong, H.Y. and Sharif, J.M. 2010. The design and implementation of a pairwise strategy supporting constraints and seeding mechanism. *Journal of Telecommunication, Electronic and Computer Engineering*. **2**(2): 1-11.

Ziyuan, W., Baowen, X. and Changhai, N. 2008. Greedy heuristic algorithms to generate variable strength combinatorial test suite. *The Eighth International Conference on Quality Software, 2008*, pp. 155-160.

APPENDICES

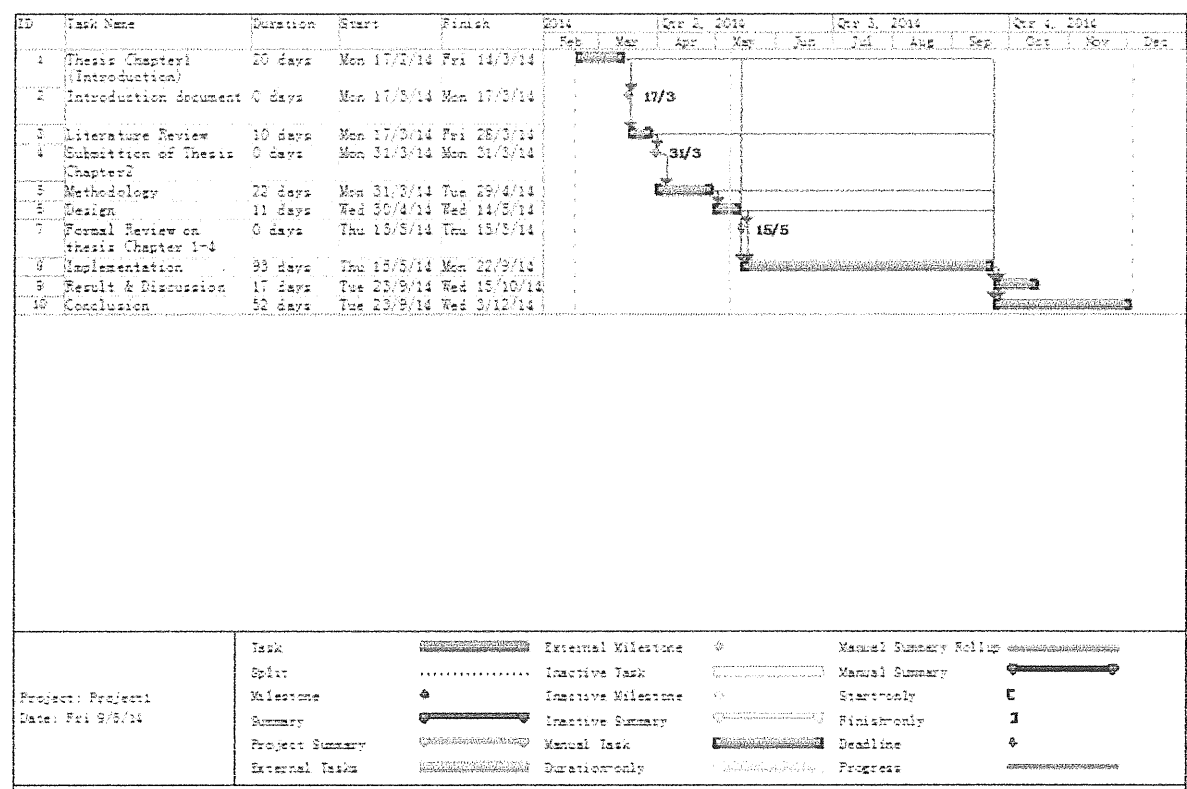


Figure 7.1: Gantt Chart


```

##### Pair List #####
PairList 0 = [0, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 1 = [0, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 2 = [0, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 3 = [1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 4 = [1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 5 = [1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 6 = [2, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 7 = [2, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 8 = [2, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 9 = [0, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 10 = [0, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 11 = [0, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 12 = [1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 13 = [1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 14 = [1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 15 = [2, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 16 = [2, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 17 = [2, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 18 = [0, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 19 = [0, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 20 = [0, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 21 = [1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 22 = [1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 23 = [1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 24 = [2, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 25 = [2, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1]

```

Figure 7.2: List of pairwise generated (13 parameter with 3 value)

```

PairList 26 = [2, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 27 = [0, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 28 = [0, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 29 = [0, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 30 = [1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 31 = [1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 32 = [1, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 33 = [2, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 34 = [2, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 35 = [2, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 36 = [0, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1]
PairList 37 = [0, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1]
PairList 38 = [0, -1, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1]
PairList 39 = [1, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1]
PairList 40 = [1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1]
PairList 41 = [1, -1, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1]
PairList 42 = [2, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1]
PairList 43 = [2, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1]
PairList 44 = [2, -1, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1]
PairList 45 = [0, -1, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1]
PairList 46 = [0, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1]
PairList 47 = [0, -1, -1, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1]
PairList 48 = [1, -1, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1]
PairList 49 = [1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1]
PairList 50 = [1, -1, -1, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1]
PairList 51 = [2, -1, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1]
PairList 52 = [2, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1]
PairList 53 = [2, -1, -1, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1]
PairList 54 = [0, -1, -1, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1]
PairList 55 = [0, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1]

```

Figure 7.3: continue

```

PairList 56 = [0, -1, -1, -1, -1, -1, -1, 2, -1, -1, -1, -1, -1]
PairList 57 = [1, -1, -1, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1]
PairList 58 = [1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1]
PairList 59 = [1, -1, -1, -1, -1, -1, -1, 2, -1, -1, -1, -1, -1]
PairList 60 = [2, -1, -1, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1]
PairList 61 = [2, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1]
PairList 62 = [2, -1, -1, -1, -1, -1, -1, 2, -1, -1, -1, -1, -1]
PairList 63 = [0, -1, -1, -1, -1, -1, -1, -1, 0, -1, -1, -1, -1]
PairList 64 = [0, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1]
PairList 65 = [0, -1, -1, -1, -1, -1, -1, -1, 2, -1, -1, -1, -1]
PairList 66 = [1, -1, -1, -1, -1, -1, -1, -1, 0, -1, -1, -1, -1]
PairList 67 = [1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1]
PairList 68 = [1, -1, -1, -1, -1, -1, -1, -1, 2, -1, -1, -1, -1]
PairList 69 = [2, -1, -1, -1, -1, -1, -1, -1, 0, -1, -1, -1, -1]
PairList 70 = [2, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1]
PairList 71 = [2, -1, -1, -1, -1, -1, -1, -1, 2, -1, -1, -1, -1]
PairList 72 = [0, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1, -1, -1]
PairList 73 = [0, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1]
PairList 74 = [0, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1, -1, -1]
PairList 75 = [1, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1, -1, -1]
PairList 76 = [1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1]
PairList 77 = [1, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1, -1, -1]
PairList 78 = [2, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1, -1, -1]
PairList 79 = [2, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1]
PairList 80 = [2, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1, -1, -1]
PairList 81 = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1, -1]
PairList 82 = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1]
PairList 83 = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1, -1]
PairList 84 = [1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1, -1]

```

Figure 7.4: continue

```

PairList 85 = [1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1]
PairList 86 = [1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1, -1]
PairList 87 = [2, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1, -1]
PairList 88 = [2, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1]
PairList 89 = [2, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1, -1]
PairList 90 = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1]
PairList 91 = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1]
PairList 92 = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1]
PairList 93 = [1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1]
PairList 94 = [1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1]
PairList 95 = [1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1]
PairList 96 = [2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1]
PairList 97 = [2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1]
PairList 98 = [2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1]
PairList 99 = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0]
PairList 100 = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1]
PairList 101 = [0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2]
PairList 102 = [1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0]
PairList 103 = [1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1]
PairList 104 = [1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2]
PairList 105 = [2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0]
PairList 106 = [2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1]
PairList 107 = [2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2]
PairList 108 = [-1, 0, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 109 = [-1, 0, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 110 = [-1, 0, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 111 = [-1, 1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 112 = [-1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 113 = [-1, 1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 114 = [-1, 2, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]

```

Figure 7.5: continue

```

PairList 115 = [-1, 2, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 116 = [-1, 2, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 117 = [-1, 0, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 118 = [-1, 0, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 119 = [-1, 0, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 120 = [-1, 1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 121 = [-1, 1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 122 = [-1, 1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 123 = [-1, 2, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 124 = [-1, 2, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 125 = [-1, 2, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 126 = [-1, 0, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 127 = [-1, 0, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 128 = [-1, 0, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 129 = [-1, 1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 130 = [-1, 1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 131 = [-1, 1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 132 = [-1, 2, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 133 = [-1, 2, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 134 = [-1, 2, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 135 = [-1, 0, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1]
PairList 136 = [-1, 0, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1]
PairList 137 = [-1, 0, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1]
PairList 138 = [-1, 1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1]
PairList 139 = [-1, 1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1]
PairList 140 = [-1, 1, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1]
PairList 141 = [-1, 2, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1]
PairList 142 = [-1, 2, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1]
PairList 143 = [-1, 2, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1]
PairList 144 = [-1, 0, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1]
PairList 145 = [-1, 0, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1]

```

Figure 7.6: continue

```

PairList 115 = [-1, 2, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 116 = [-1, 2, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 117 = [-1, 0, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 118 = [-1, 0, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 119 = [-1, 0, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 120 = [-1, 1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 121 = [-1, 1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 122 = [-1, 1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 123 = [-1, 2, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 124 = [-1, 2, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 125 = [-1, 2, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 126 = [-1, 0, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 127 = [-1, 0, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 128 = [-1, 0, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 129 = [-1, 1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 130 = [-1, 1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 131 = [-1, 1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 132 = [-1, 2, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 133 = [-1, 2, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 134 = [-1, 2, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1, -1]
PairList 135 = [-1, 0, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1]
PairList 136 = [-1, 0, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1]
PairList 137 = [-1, 0, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1]
PairList 138 = [-1, 1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1]
PairList 139 = [-1, 1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1]
PairList 140 = [-1, 1, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1]
PairList 141 = [-1, 2, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1]
PairList 142 = [-1, 2, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1]
PairList 143 = [-1, 2, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1, -1]
PairList 144 = [-1, 0, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1]
PairList 145 = [-1, 0, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1]

```

Figure 7.7: *continue*

```

PairList 671 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, 2]
PairList 672 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1, -1, 0]
PairList 673 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1, -1, 1]
PairList 674 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1, -1, 2]
PairList 675 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 0, -1]
PairList 676 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 1, -1]
PairList 677 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 2, -1]
PairList 678 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 0, -1]
PairList 679 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1, -1]
PairList 680 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 2, -1]
PairList 681 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, 0, -1]
PairList 682 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, 1, -1]
PairList 683 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, 2, -1]
PairList 684 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1, 0]
PairList 685 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1, 1]
PairList 686 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1, 2]
PairList 687 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, 0]
PairList 688 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, 1]
PairList 689 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, 2]
PairList 690 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1, 0]
PairList 691 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1, 1]
PairList 692 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, -1, 2]
PairList 693 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 0]
PairList 694 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 1]
PairList 695 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 2]
PairList 696 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 0]
PairList 697 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1]
PairList 698 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 2]
PairList 699 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, 0]
PairList 700 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, 1]
PairList 701 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 2, 2]

```

Figure 7.8 : continue

```

***** List *****
Case 0 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Case 1 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
Case 2 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2]
Case 3 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
Case 4 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
Case 5 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2]
Case 6 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0]
Case 7 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1]
Case 8 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2]
Case 9 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
Case 10 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1]
Case 11 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2]
Case 12 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0]
Case 13 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1]
Case 14 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2]
Case 15 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0]
Case 16 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1]
Case 17 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 2]
Case 18 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0]
Case 19 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1]
Case 20 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2]
Case 21 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0]
Case 22 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1]
Case 23 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 2]
Case 24 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0]
Case 25 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 1]
Case 26 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2]
Case 27 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
Case 28 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
Case 29 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2]

```

Figure 7.9 : list of possible combination (13 parameter with 3 value)


```

case 1594293 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 0]
case 1594294 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1]
case 1594295 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2]
case 1594296 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0]
case 1594297 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 1]
case 1594298 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2]
case 1594299 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 0]
case 1594300 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1]
case 1594301 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 2]
case 1594302 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0]
case 1594303 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 1]
case 1594304 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2]
case 1594305 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0, 0]
case 1594306 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0, 1]
case 1594307 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0, 2]
case 1594308 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 0]
case 1594309 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1]
case 1594310 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2]
case 1594311 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 0]
case 1594312 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1]
case 1594313 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2]
case 1594314 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0]
case 1594315 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1]
case 1594316 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2]
case 1594317 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0]
case 1594318 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1]
case 1594319 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2]
case 1594320 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0]
case 1594321 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1]
case 1594322 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]

```

Figure 7.10: continue

```

***** Result *****
0 Best= [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
1 Best= [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
2 Best= [0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
3 Best= [1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2]
4 Best= [1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 0, 0, 0]
5 Best= [1, 2, 2, 2, 2, 0, 0, 0, 0, 0, 1, 1, 1]
6 Best= [2, 0, 0, 0, 0, 2, 2, 2, 2, 2, 1, 1, 1]
7 Best= [2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 2, 2, 2]
8 Best= [2, 2, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0, 0]
9 Best= [2, 2, 2, 2, 2, 0, 1, 1, 1, 1, 0, 2, 0]
10 Best= [2, 0, 1, 2, 2, 1, 0, 2, 1, 2, 0, 1, 0]
11 Best= [2, 2, 2, 0, 1, 1, 2, 0, 1, 0, 1, 2, 0]
12 Best= [0, 1, 0, 2, 2, 2, 1, 1, 0, 0, 2, 1, 0]
13 Best= [0, 2, 2, 1, 0, 2, 0, 0, 1, 0, 2, 0, 1]
14 Best= [0, 0, 0, 1, 1, 0, 0, 1, 2, 1, 1, 0, 2]
15 Best= [0, 1, 1, 0, 0, 0, 2, 2, 0, 2, 0, 0, 1]
16 Best= [0, 0, 1, 2, 0, 1, 1, 0, 2, 0, 0, 1, 2]
17 Best= [0, 0, 2, 0, 1, 0, 1, 2, 1, 0, 0, 2, 1]
18 Best= [0, 1, 2, 0, 0, 0, 1, 0, 2, 1, 2, 1, 2]
19 Best= [0, 2, 0, 0, 2, 0, 2, 1, 1, 1, 2, 0, 0]
20 Best= [0, 2, 1, 0, 1, 1, 0, 0, 0, 1, 2, 0, 0]
21 Best= [0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 2, 0, 0]
22 Best= [0, 0, 0, 1, 2, 0, 0, 0, 0, 2, 1, 0, 0]
23 Best= [0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Uncover= 0
BUILD SUCCESSFUL (total time: 24 minutes 38 seconds)

```

Figure 7.11 : the result (13 parameter with 3 value)

```

run:
10, 10, 10, 10, 10, 10, 10, 10, 10, 10

Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at pkg3.Run.generate(Run.java:33)
    at pkg3.Run.analyse(Run.java:20)
    at pkg3.Main.main(Main.java:17)
Java Result: 1
BUILD SUCCESSFUL (total time: 27 seconds)

```

Figure 7.12: failure print screen during extreme huge data set

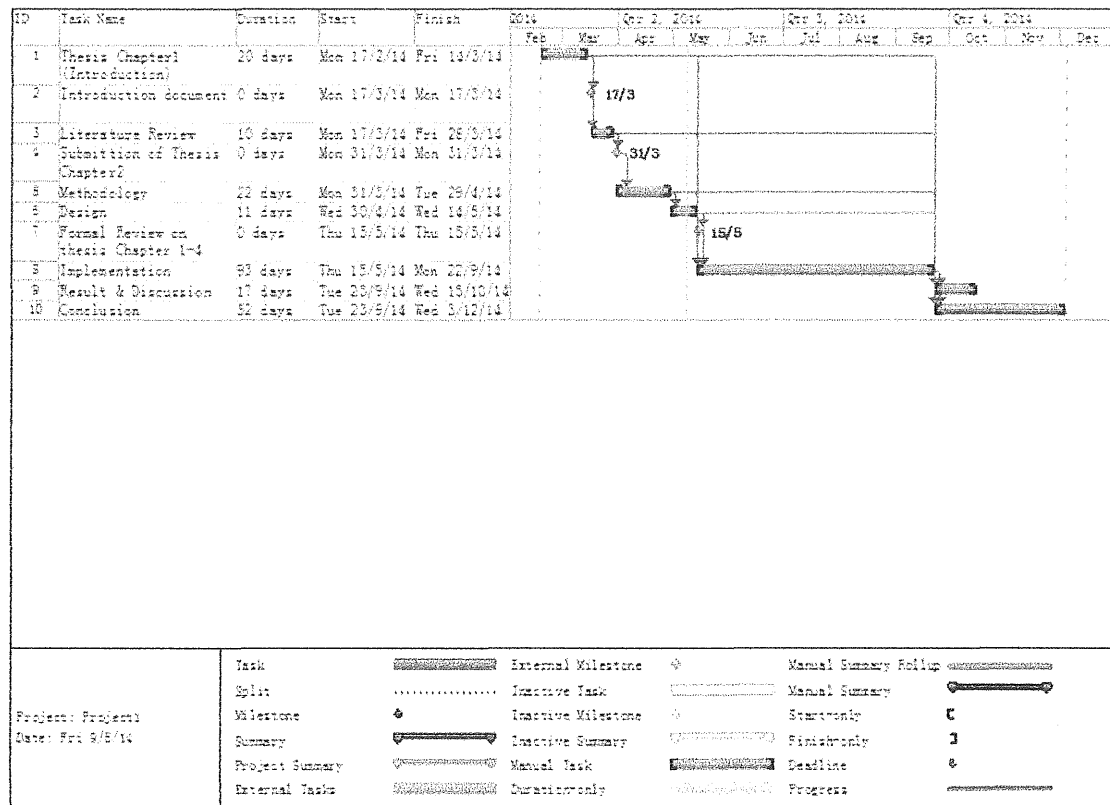


Figure 7.13: Gantt chart