

**CTJ: INPUT-OUTPUT BASED RELATION
COMBINATORIAL TESTING STRATEGY
USING JAYA ALGORITHM**

NG YEONG KHANG

**BACHELOR OF COMPUTER SCIENCE
(SOFTWARE ENGINEERING)**

UNIVERSITI MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name : Ng Yeong Khang

Date of Birth : 19/11/1995

Title : CTJ: Input-Output Based Relation Combinatorial Testing Strategy
Using Jaya Algorithm

Academic Session : 2018/2019

I declare that this thesis is classified as:

- CONFIDENTIAL (Contains confidential information under the Official Secret Act 1997)*
- RESTRICTED (Contains restricted information as specified by the organization where research was done)*
- OPEN ACCESS I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang reserves the following rights:

1. The Thesis is the Property of Universiti Malaysia Pahang
2. The Library of Universiti Malaysia Pahang has the right to make copies of the thesis for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:

(Student's Signature)

Ng Yeong Khang
951119-07-5055
Date: 08/01/2019

(Supervisor's Signature)

Dr. AbdulRahman al-Sewari
Date: 08/01/2019



SUPERVISOR'S DECLARATION

I hereby declare that I have checked this thesis and in my opinion, this thesis is adequate in terms of scope and quality for the award of the degree of Bachelor of Computer Science (Software Engineering).

(Supervisor's Signature)

Full Name : Dr. AbdulRahman al-Sewari

Position : Senior Lecturer

Date : 8 January 2019

STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

(Student's Signature)

Full Name : Ng Yeong Khang

ID Number : CB15092

Date : 8 January 2019

CTJ: INPUT-OUTPUT BASED RELATION COMBINATORIAL TESTING
STRATEGY USING JAYA ALGORITHM

NG YEONG KHANG

Thesis submitted in fulfillment of the requirements
for the award of the degree of
Bachelor of Computer Science (Software Engineering)

Faculty of Computer Systems and Software Engineering
UNIVERSITI MALAYSIA PAHANG

JANUARY 2019

ACKNOWLEDGEMENTS

First, I would express my deepest appreciation to my respected final year project supervisor, Dr. AbdulRahman al-Sewari in guiding me to complete this thesis which has the title of CTJ: Input-Output Based Relation Combinatorial Testing Strategy Using Jaya Algorithm. He always welcomes me to consult him anytime whenever I faced any problem in doing this final year project. His expertise in software testing gave me insight and well understanding on combinatorial testing and this knowledge accelerates my progress in completing this thesis.

Moreover, I would like to thank to my family members especially my dear parents in supporting me throughout the implementation of my final year project. They always give me advice and backing when I was no idea to solve the trouble encountered. The strong mental support given by them is a great motivation for me to complete this thesis.

Furthermore, a special thanks goes to all friends of mine who gave their helping hands when I need any assistance from them. Their willingness in helping me should be acknowledged as their contributions made the progress of my final year project run uneventful.

Finally, a special gratitude I want to give to all lecturers and staff in Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang that helped me directly and indirectly in completing this final year project. Their precious helps made me able to work smoothly throughout this final year project.

ABSTRAK

Pengujian perisian adalah salah satu unsur yang penting dalam pembangunan perisian. Kebanyakan masa, sistem yang diuji mempunyai lebih daripada satu input dan pengujian setiap kombinasi input adalah hampir mustahil kerana masa pelaksanaan kes ujian terlalu panjang. Pengujian kombinatorial ialah satu cara untuk menggantikan ujian menyeluruh melalui pengujian setiap nilai input dan setiap kombinasi antara parameter. Pengujian kombinatorial boleh dibahagikan kepada tiga jenis iaitu interaksi kekuatan seragam, interaksi kekuatan berubah-ubah dan hubungan berdasarkan input-output (IOR). Pengujian kombinatorial IOR hanya menguji kombinasi penting yang dipilih oleh penguji. Kebanyakan penyelidikan dalam pengujian kombinatorial menggunakan interaksi kekuatan seragam dan berubah-ubah tetapi terdapat hanya beberapa kajian yang menangani IOR. Oleh hal sedemikian, pengujian kombinatorial IOR dipilih untuk dikaji dalam kajian ini. Untuk mengatasi masalah pengoptimalan gabungan, algoritma Jaya dicadangkan untuk digunakan dalam projek ini disebabkan algoritma metaheuristik pantas dalam pengoptimuman dan strategi ini dinamakan sebagai CTJ. Hasil penerapan algoritma Jaya dalam pengujian kombinatorial input-output dapat diterima kerana menghasilkan jumlah kes ujian yang hampir optimum dalam tempoh masa yang memuaskan.

ABSTRACT

Software testing is a vital part in software development lifecycle. Most of the time, system under test has more than one input and testing of every combinations of inputs is almost impossible as the time of execution of test case is outrageously long. Combinatorial testing is the way to encounter exhaustive testing through the testing of every input values and every combination between parameters. Combinatorial testing can be divided into three types which are uniform strength interaction, variable strength interaction and input-output based relation (IOR). IOR combinatorial testing only test for the important combinations that selected by tester. Most of the researches in combinatorial testing applied uniform and variable interaction strength but there are only few studies feature IOR. Thus, IOR combinatorial testing is selected to be studied in this research. To overcome the combinatorial optimization problem, Jaya algorithm is proposed to apply in this project since metaheuristic algorithm is fast in optimization and this strategy is named as CTJ. The result of applying Jaya algorithm in input-output based combinatorial testing is acceptable since it produces nearly optimum number of test cases in the satisfactory time range.

TABLE OF CONTENT

ACKNOWLEDGEMENTS	vi
ABSTRAK	vii
ABSTRACT	viii
TABLE OF CONTENT	ix
LIST OF TABLES	xii
LIST OF FIGURES	xiii
LIST OF ABBREVIATIONS	xv
CHAPTER 1 INTRODUCTION	1
1.1 INTRODUCTION	1
1.2 PROBLEM STATEMENT	2
1.3 AIM AND OBJECTIVES	5
1.4 SCOPE	6
1.5 THESIS ORGANIZATION	6
CHAPTER 2 LITERATURE REVIEW	7
2.1 INPUT-OUTPUT BASED RELATION COMBINATORIAL TESTING	7
2.2 RELATED WORK	12
2.2.1 Pure Computational Approach	12
2.2.2 Natural Based Approach	18
2.3 JAYA ALGORITHM	27
CHAPTER 3 METHODOLOGY	32
3.1 INTRODUCTION	32

3.2	METHODOLOGY	32
3.2.1	Phase 1: Literature Review	33
3.2.2	Phase 2: Design the Solution	34
3.2.3	Phase 3: Implementation of the Solution	42
3.2.4	Phase 4: Test and Evaluation	51
3.2.5	Phase 5: Documentation	55
3.3	HARDWARE AND SOFTWARE	55
3.4	GANTT CHART	56
	CHAPTER 4 IMPLEMENTATION, RESULT AND DISCUSSION	57
4.1	INTRODUCTION	57
4.2	IMPLEMENTATION OF CTJ	57
4.2.1	Level 1: Reading of input values	57
4.2.2	Level 2: Data analysis and data mapping	60
4.2.3	Level 3: Combinations of input values generation	61
4.2.4	Level 4: Test case generation based on Jaya algorithm	62
4.2.5	Level 5: Finalization of test suite generation	64
4.3	EXPERIMENTAL RESULTS AND DISCUSSION	64
4.3.1	Parameter tuning of CTJ	64
4.3.2	Experiments for input-output based relation	66
4.3.3	Experiments for uniform interaction strength	67
	CHAPTER 5 CONCLUSION	70
5.1	INTRODUCTION	70
5.2	RESEARCH CONSTRAINTS	71
5.3	FUTURE WORKS	71

REFERENCES	72
APPENDIX A GANTT CHART	77

LIST OF TABLES

Table 1.1 All possible input values for “Print” section in Notepad++’s Preferences	4
Table 2.1 All input parameters and their input values	8
Table 2.2 List of all possible combinations of input values with $t = 2$	9
Table 2.3 Test suite generated through pairwise combinatorial testing	9
Table 2.4 List of all possible combinations of input values with IOR	11
Table 2.5 Test suite generated through IOR	12
Table 2.6 Comparison between natural based approaches	18
Table 2.7 Comparison between natural based approaches	27
Table 2.8 Comparison between Jaya algorithm with existing strategies	31
Table 3.1 Result of data mapping using the input values from Table 2.1	35
Table 3.2 60 input-output relationships (R) that utilized in experiments	52
Table 3.3 The size of test suite of existing IOR strategies using configuration IOR (N, 3^{10} , R)	52
Table 3.4 The size of test suite of existing IOR strategies using configuration (N, 2^3 3^3 4^3 5^1 , R)	53
Table 3.5 System configuration for uniform interaction strength experiment	54
Table 3.6 The test suite size of existing strategies using uniform interaction strength	54
Table 3.7 List of needed hardware	55
Table 3.8 List of needed software	56
Table 4.1 The mapped values for both input parameters and their corresponding values	60
Table 4.2 Parameter Setting	64
Table 4.3 Test case size and execution time in 30 input-output relationships configuration	65
Table 4.4 Test case size and execution time in CA(N; 3, 6, 6) configuration	65
Table 4.5 Test case size and execution time of IOR (N, 3^{10} , R) configuration in first IOR experiment	68
Table 4.6 Test case size and execution time of IOR (N, 2^3 , 3^3 , 4^3 , 5^1 , R) configuration in second IOR experiment	68
Table 4.7 Test case size and execution time of different configurations in uniform interaction strength experiment	69

LIST OF FIGURES

Figure 1.1 Print section of Preferences of Notepad ++	3
Figure 2.1 Input-output relationship between P, Q, R, S and X, Y, Z of Program P1	11
Figure 2.2 Example to show Greedy algorithm's problem	13
Figure 2.3 Pseudocode of recursive Greedy algorithm (Cormen et al., 2009)	14
Figure 2.4 Pseudocode of test case generation using concept of Density (Z. Wang et al., 2008)	15
Figure 2.5 Overview of AURA strategy (Ong & Zamli, 2011)	16
Figure 2.6 Pseudocode of interaction pair generation algorithm (Ong & Zamli, 2011)	17
Figure 2.7 Pseudocode of test suite generation algorithm (Ong & Zamli, 2011)	17
Figure 2.8 Pseudocode of actual data mapping algorithm (Ong & Zamli, 2011)	18
Figure 2.9 Pseudocode of Ant Colony Optimization (Blum, 2005)	20
Figure 2.10 The illustration on how ants find the shortest path between food source and their nest (Blum, 2005)	20
Figure 2.11 Crossover operation to generate offspring (Elbeltagi, Hegazy, & Grierson, 2005)	22
Figure 2.12 Pseudocode for Genetic Algorithm (Elbeltagi et al., 2005)	22
Figure 2.13 Pseudocode of Harmony Search algorithm (Abdul Rahman, 2012)	24
Figure 2.14 Pseudocode of Particle Swarm Optimization (Poli et al., 2007)	25
Figure 2.15 Pseudocode of Simulated Annealing algorithm (Xambre & Vilarinho, 2003)	26
Figure 2.16 Flowchart of Jaya Algorithm (R Rao, 2016)	29
Figure 3.1 The flowchart of research methodology	33
Figure 3.2 The flowchart of the execution of IOR combinatorial testing based on Jaya algorithm	34
Figure 3.3 The flowchart of test case generation using Jaya algorithm	37
Figure 3.4 GUI for the input of parameters and its values	38
Figure 3.5 GUI of load from file	39
Figure 3.6 GUI of uniform interaction strength	40
Figure 3.7 GUI of input-output based relation	41
Figure 3.8 Pseudocode on reading the parameters and their corresponding values	42
Figure 3.9 Pseudocode of reading all necessary information for test case generation	43
Figure 3.10 Pseudocode for data analysis	43
Figure 3.11 Pseudocode for data mapping	44

Figure 3.12 Pseudocode of combination of input values generation for uniform strength interaction	46
Figure 3.13 Pseudocode of combination of input values generation for input-output relationship	47
Figure 3.14 Pseudocode for test case generation using Jaya algorithm (Part 1)	49
Figure 3.15 Pseudocode for test case generation using Jaya algorithm (Part 2)	50
Figure 3.16 Pseudocode of finalization of test case generation	51
Figure 4.1 Example of completed input parameters and its corresponding values at the Home page of CTJ	58
Figure 4.2 File selection for Load From File option	59
Figure 4.3 The GUI after reading the data from file	59
Figure 4.4 Error message for duplication of value entered	60
Figure 4.5 The complete details that have to be filled in before test case generation through uniform interaction strength	61
Figure 4.6 The complete details that have to be filled in before test case generation through input-output relationships	62
Figure 4.7 The test suite generated through ordinary GUI input	63
Figure 4.8 Test suite generated through Load From File	63

LIST OF ABBREVIATIONS

ACO	Ant Colony Optimization
AETG	Automatic Efficient Test Generator
AGC	Controller for Automatic Generation Control
AI	Artificial Intelligence
CA	Covering Array
EA	Evolutionary Algorithm
FIFT	Failure-Triggering Fault Interaction
GA	Genetic Algorithm
GUI	Graphical User Interface
HM	Harmony Memory
HMCR	Harmony Memory Considering Rate
HMS	Harmony Memory Size
HS	Harmony Search
IOR	Input-Output Based Relation
MCA	Mixed Level Covering Array
PAR	Pitch Adjustment Rate
PHSS	Pairwise Harmony Search Testing Strategy
PID	Proportional-Integral-Derivative
PSO	Particle Swarm Optimization
PSTG	Particle Swarm Test Generator
PV-DSTAT-COM	Photovoltaic Fed Distributed Static Compensator
QAP	Quadratic Assignment Problem
SA	Simulated Annealing
SBPWM	Simple Boost Pulse Width Modulation
SI	Swarm Intelligence
SUT	System Under Test
TLBO	Teaching-Learning-Based Optimization

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Software testing is an inevitable process in software development lifecycle to find out the software bugs by validating and verifying the application whether it works as expected and meets the business and technical requirements. A recent report from Tricentis, a leading software testing company in Continuous Testing found that there are 606 recorded software failure that happened around the globe which affected over 3.7 billion people and 314 companies as well as \$1.7 trillion in lost revenue and 268 years of downtime (Tricentis, 2018). Therefore, a more effective defect detection approach needed to be carried out to increase the coverage of testing.

Combinatorial testing is a black-box testing technique that generate test cases by combining the values of different test object input parameters using combinatorial optimization strategies (De Vries, Vohra, Economics, & Science, 2003). Taking the study from the failure of medical device application, the failure-triggering fault interaction (FTFI) is 68% for single parameter value, 97% of failures triggered by 2 combination values while the percentage of failures caused by 3 and 4 combination values are 99% and 100% respectively (Kuhn, Wallace, & Gallo, 2004). By using combinatorial testing, all input values of the test objects and interactions between each parameter are tested which result in higher detection of interaction failure compared to single parameter testing.

Combinatorial optimization is a process of searching the optimum number of test cases for combinatorial testing. There are many different optimization strategies that are used to generate the test cases for combinatorial testing such as Harmony Search (A. R. A. Alsewari & Zamli, 2012), Genetic Algorithm (Shiba, Tsuchiya, & Kikuno, 2004b),

Ant Colony Algorithm (Shiba et al., 2004b), Simplified Swarm Optimization (Ahmed, Sahib, & Potrus, 2014), Differential Evolution Algorithm (Liang, Guo, Huang, & Jiao, 2014) and so on. Jaya Algorithm is chosen to be applied in this study as this algorithm has been used in lots of optimization problems in other fields.

Combinatorial testing also known as interaction t-way testing where t represents the interaction strength. There are two types of t-way interaction which are uniform strength t-way interaction and variable strength t-way interaction. The interaction between all parameters are uniform in uniform strength t-way interaction while variable strength t-way interaction involves main uniform interaction and sub-uniform interaction. Both type of interactions will generate all possible interactions between each parameter. Often, some of the interactions generated maybe not even be used in the testing. This waste the precious time and effort of the tester to generate those useless interactions. Hence, input-output based relation (IOR) has been introduced in combinatorial optimization to improve the efficiency in finding optimum number of test case as well as given the flexibility in selecting the desired parameter and its interaction (A. R. A. Alsewari & Zamli, 2012).

1.2 PROBLEM STATEMENT

In most of the software application, often there exists one part of system input required to enter a combination of values or choices. The system under test (SUT) is then needed to test for every combination of input parameter to make sure the actual behaviour of the system is same as expected behaviour since the cost of fixing the defect found after software delivered is much higher. Testing of each combination of values is a time and effort wasting job and this leads to exhaustive testing. Exhaustive testing is an impractical software testing technique and usually impossible to achieve in the real testing environment due to budget available and time constraint to execute all combinations of inputs.

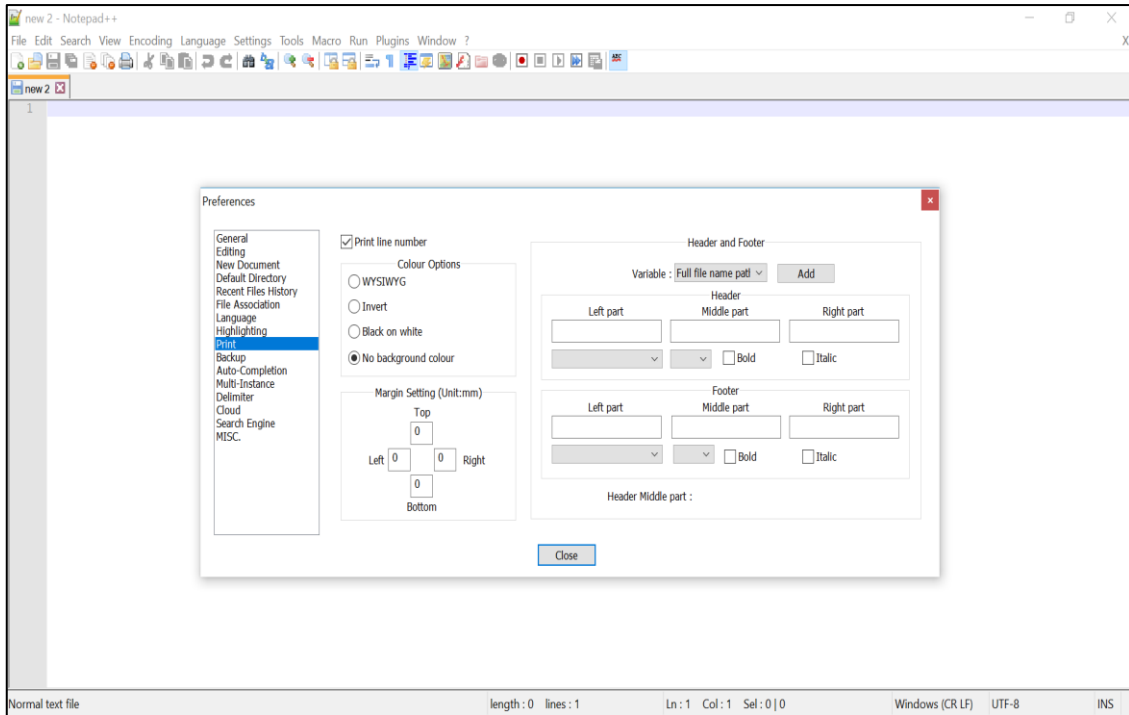


Figure 1.1 Print section of Preferences of Notepad ++

Taking the example from the renowned open source code editor, Notepad ++, the “Print” section in the Preferences as shown in Figure 1 is chosen to show the total number of test cases needed to carry out the testing process via exhaustive testing. There are 22 input parameters required to enter in the “Print” section and all possible input values are shown in Table 1.

Table 1.1 All possible input values for “Print” section in Notepad++’s Preferences

Input parameter	Input values/ Range	Number of possible inputs
Print line number	Check, Unchecked	2
Colour options	WYSIWYG, Invert, Black on white, No background colour	4
Margin Setting		
Top	0 to 99	100
Bottom	0 to 99	100
Left	0 to 99	100
Right	0 to 99	100
Variable	Full file name path, File name, File directory, Page, Short date format, Long date format, Time	7
Header		
Left part	Minimum number of characters is 0 while maximum number of characters is 50	51
Middle part	Minimum number of characters is 0 while maximum number of characters is 50	51
Right part	Minimum number of characters is 0 while maximum number of characters is 50	51
Font	276 types of fonts	276
Font size	6 to 14	9
Bold	Check, Unchecked	2
Italic	Check, Unchecked	2
Footer		
Left part	Minimum number of characters is 0 while maximum number of characters is 50	51
Middle part	Minimum number of characters is 0 while maximum number of characters is 50	51
Right part	Minimum number of characters is 0 while maximum number of characters is 50	51
Font	276 types of fonts	276
Font size	6 to 14	9
Bold	Check, Unchecked	2
Italic	Check, Unchecked	2

Based on the number of possible inputs in Table 1, the total number of test cases required for exhaustive testing that cover all possible interactions between each input value are 9,728,194,594,213,496,217,600,000,000 which approximately to 9 octillions. The calculation of total number of test cases is done by multiplying all possible number of input values. Given the execution time of each test case is 1 second, the total time taken to complete all test case execution is 308,479,026,960,093,106,849.32 years (nearly 308 quintillion years) which exceeds human lifetime to complete the testing for one section in the Preferences in Notepad++.

Moreover, the interactions generated through uniform strength or variable strength in combinatorial testing mostly are not fully utilized in real testing environment. This may cause the important interactions between the input parameters are excluded and unwanted test cases are including in the test suite which make the size of test suite increases (Ramli, Othman, & Ali, 2016). Wasting of time to generate the unrelated interactions and test cases is happened as well as fail to reduce the test suite's redundancy. The increase of the number of test cases in test suite will increase the time to find out the fault in the test object and this will induce the rise of testing effort.

To overcome the exhaustive testing and reduce the number of unwanted test cases produced by combinatorial testing, input-output based relation combinatorial testing strategy based on Jaya Algorithm which known as CTJ is introduced in this study.

1.3 AIM AND OBJECTIVES

The goal of this thesis is to implement input-output based relation combinatorial testing strategy using Jaya Algorithm (CTJ). Several objectives have been identified to accomplish to succeed the goal of this thesis:

- I. To study the existing input-output based relation combinatorial testing strategies.
- II. To implement Jaya Algorithm in input-output based combinatorial testing.
- III. To assess the performance of the proposed combinatorial testing strategy.

1.4 SCOPE

The scopes of this project are stated as below:

- I. The program is built based on Java programming language.
- II. The program is worked on desktop platform only.
- III. The degree of interaction strength of combinatorial testing is only two, three and four.

1.5 THESIS ORGANIZATION

This thesis is made up of five chapter. Chapter 1 is about the overview of the project which consists of background of study, problem statement, aim & objectives and scope of the project. Next, Chapter 2 is the literature review of existing input-output based relation combinatorial testing metaheuristic algorithm, comparison between each metaheuristic algorithm and Jaya Algorithm. Chapter 3 is the methodology of CTJ which including the hardware and software needed, Gantt chart as well as the implementation flow and testing design. Further, the implementation, result of execution and discussion on the performance of CTJ are in Chapter 4. Lastly, Chapter 5 is the conclusion for this thesis that includes summary, research constraints and future works of CTJ.

CHAPTER 2

LITERATURE REVIEW

2.1 INPUT-OUTPUT BASED RELATION COMBINATORIAL TESTING

Combinatorial testing also known as t-way interaction testing need every t-way combination of input parameters and its values to be covered in at least one test case in the test suite (Shiba et al., 2004b). This approach can be divided into three main types which are uniform interaction strength, variable interaction strength and input-output based relation. Uniform interaction strength and variable interaction strength rely on the interaction strength (t) which used to decide the interactions between each input parameters. Pairwise testing where t is equal to two is one of the combinatorial technique that has the interaction between the input parameters in the pair form (McCaffrey, 2009). In the other word, the input parameters will be paired up with each other and the test suite is generated based on the combination of input parameters' values which covered by at least one of the test cases.

Combinatorial testing can be expressed into a covering array (CA). CA is widely implemented in interaction testing where all input parameters have equal number of input values to generate a test suite that cover all possible interactions. *A covering array, $CA_{\lambda}(N; t, k, v)$, is an $N \times k$ array on v symbols such that every $N \times t$ sub-array contains all ordered subsets from v symbols of size t at least λ times* (Myra B. Cohen, 2004). In the simple word, CA is an array with N^{th} row and k^{th} column which satisfies the condition that all t-tuples are covered in these rows at least once. N represents the number of test cases, t is the interaction strength while k and v are the number of input parameters and the number of values for each input parameters. The value of t must be the same for all parameters since this is a uniform interaction strength combinatorial testing.

Mixed level covering array (MCA) is another mathematical way to present the combinatorial testing for the input parameters that have different number of input values. A mixed level covering array, $MCA_{\lambda}(N; t, k, (v_1, v_2, \dots, v_k))$, is an $N \times k$ array on v symbols, where $v = \sum_{x=1}^k v_x$, with the following properties:

1. Each column i ($1 \leq i \leq k$) contains only elements from a set S_i of size v_i .
2. The rows of each $N \times t$ sub-array covers all t -tuples of values from the t columns at least λ times.

Above is the definition of mixed level covering array from the research of (Myra B. Cohen, 2004). From the formula on the above, (v_1, v_2, \dots, v_k) represent the number of input values for specific input parameter respectively. The only difference between the CA and MCA is the input parameter of MCA can have different number of input values while every input parameter in CA must have same amount of input values.

The uniform interaction strength combinatorial testing is demonstrated as below. Table 2.1 shows all input parameters and its input values which will be involved in generating the optimum number of test cases.

Table 2.1 All input parameters and their input values

Parameters	P	Q	R	S
Input values	P ₁	Q ₁	R ₁	S ₁
	P ₂	Q ₂	R ₂	S ₂
	P ₃	Q ₃		S ₃
		Q ₄		

MCA is chosen to represent this model since not every parameter are having the same number of input values. The MCA formula for this model is $MCA(N, 2, 4, (3, 4, 2, 3))$. N is still unknown as the number of test cases needed is not yet compute. All parameters and their input values are then paired up to come out with a list of combinations of input values which is presented in Table 2.2.

Table 2.2 List of all possible combinations of input values with t = 2

PQ	PR	PS	QR	QS	RS
P ₁ Q ₁	P ₁ R ₁	P ₁ S ₁	Q ₁ R ₁	Q ₁ S ₁	R ₁ S ₁
P ₁ Q ₂	P ₁ R ₂	P ₁ S ₂	Q ₁ R ₂	Q ₁ S ₂	R ₁ S ₂
P ₁ Q ₃	P ₂ R ₁	P ₁ S ₃	Q ₂ R ₁	Q ₁ S ₃	R ₁ S ₃
P ₁ Q ₄	P ₂ R ₂	P ₂ S ₁	Q ₂ R ₂	Q ₂ S ₁	R ₂ S ₁
P ₂ Q ₁	P ₃ R ₁	P ₂ S ₂	Q ₃ R ₁	Q ₂ S ₂	R ₂ S ₂
P ₂ Q ₂	P ₃ R ₂	P ₂ S ₃	Q ₃ R ₂	Q ₂ S ₃	R ₂ S ₃
P ₂ Q ₃		P ₃ S ₁	Q ₄ R ₁	Q ₃ S ₁	
P ₂ Q ₄		P ₃ S ₂	Q ₄ R ₂	Q ₃ S ₂	
P ₃ Q ₁		P ₃ S ₃		Q ₃ S ₃	
P ₃ Q ₂				Q ₄ S ₁	
P ₃ Q ₃				Q ₄ S ₂	
P ₃ Q ₄				Q ₄ S ₃	

From the list of all possible combinations of input values in Table 2.2, the test suite is generated where all combinations are covered in one of the test case by at least one time.

Table 2.3 Test suite generated through pairwise combinatorial testing

Test Case	Input Parameters				Occurrences	Combination Covered
	P	Q	R	S		
1	P ₁	Q ₃	R ₁	S ₂	6	(P ₁ , Q ₃), (P ₁ , R ₁), (P ₁ , S ₂), (Q ₃ , R ₁), (Q ₃ , S ₂), (R ₁ , S ₂)
2	P ₃	Q ₁	R ₂	S ₃	6	(P ₃ , Q ₁), (P ₃ , R ₂), (P ₃ , S ₃), (Q ₁ , R ₂), (Q ₁ , S ₃), (R ₂ , S ₃)
3	P ₂	Q ₄	R ₂	S ₁	6	(P ₂ , Q ₄), (P ₂ , R ₂), (P ₂ , S ₁), (Q ₄ , R ₂), (Q ₄ , S ₁), (R ₂ , S ₁)
4	P ₂	Q ₂	R ₁	S ₃	6	(P ₂ , Q ₂), (P ₂ , R ₁), (P ₂ , S ₃), (Q ₂ , R ₁), (Q ₂ , S ₃), (R ₁ , S ₃)
5	P ₃	Q ₂	R ₁	S ₁	5	(P ₃ , Q ₂), (P ₃ , R ₁), (P ₃ , S ₁), (Q ₂ , S ₁), (R ₁ , S ₁)
6	P ₁	Q ₂	R ₂	S ₂	5	(P ₁ , Q ₂), (P ₁ , R ₂), (Q ₂ , R ₂), (Q ₂ , S ₂), (R ₂ , S ₂)
7	P ₃	Q ₄	R ₁	S ₂	4	(P ₃ , Q ₄), (P ₃ , S ₂), (Q ₄ , R ₁), (Q ₄ , S ₂)
8	P ₁	Q ₁	R ₁	S ₁	4	(P ₁ , Q ₁), (P ₁ , S ₁), (Q ₁ , R ₁), (Q ₁ , S ₁)
9	P ₂	Q ₁	R ₁	S ₂	3	(P ₂ , Q ₁), (P ₂ , S ₂), (Q ₁ , S ₂)
10	P ₁	Q ₄	R ₁	S ₃	3	(P ₁ , Q ₄), (P ₁ , S ₃), (Q ₄ , S ₃)
11	P ₃	Q ₃	R ₂	S ₁	3	(P ₃ , Q ₃), (Q ₃ , R ₂), (Q ₃ , S ₁)
12	P ₂	Q ₃	R ₂	S ₃	2	(P ₂ , Q ₃), (Q ₃ , S ₃)

Based on the test suite in Table 2.3, the final MCA formula for this interaction testing is $MCA(12, 2, 4, (3, 4, 2, 3))$. The number of test cases required to comprise all input values in at least one test case through 2-way combinatorial testing is 12 test cases while the number of test cases produced using exhaustive testing is 72 test cases. This shows combinatorial testing strategy significantly cut down the amount of test cases by 6 times if measured with number of test cases generated using exhaustive testing.

Combinatorial testing using uniform strength interaction does decrease the number of test cases for black box testing but not all combinations of input values are going to use in the real-world scenario. Therefore, input-output based relation (IOR) combinatorial testing has been introduced by Schroeder P.J. et al to overcome this problem by handling the input combinations that will generate the desired output instead of using all possible input combinations (Patrick J. Schroeder & Korel, 2000). Information on the relationships between the input values and output value is taking into consideration in this approach.

To express IOR in the mathematical way, a combination of input-output relationship (Rel) and covering array are needed to come out with input-output based relations covering array. Rel can be written in this form, $Rel = \{\{x_1\}, \{x_2\}, \dots, \{x_n\}\}$ where x is the combination of inputs that will generate the specific output. Input-output based relations covering array, IOR (N, C, Rel) is the mathematical form of IOR relation. N represents the number of test case in the test suite, C is the number of value of each input parameter ($v_1^{P_1}, v_2^{P_2}, \dots, v_n^{P_n}$) where v is the amount of input value and p is the amount of parameter that has the same amount of v while Rel is the input-output relationship as stated above (Othman & Zamli, 2011).

To further explain the implementation of IOR in combinatorial testing, the program P1 in the research of Schroeder P.J. et al has been adopted (Patrick J. Schroeder & Korel, 2000). Considering the input parameters from Table 2.1, the inputs (P, Q, R, S) are having three outputs (X, Y, Z). Output X is the combination between inputs P and R, output Y is the combination between inputs R and S while the output Z is the combination of inputs Q and R. In this case, the input-output relationship is $Rel (\{P, R\}, \{Q, R\} \{R, S\})$. The input-output based relations covering array for this instance is IOR ($N, 3^2 4^1 2^1, Rel$) with N is unknown since the IOR combinatorial testing have not taken place yet.

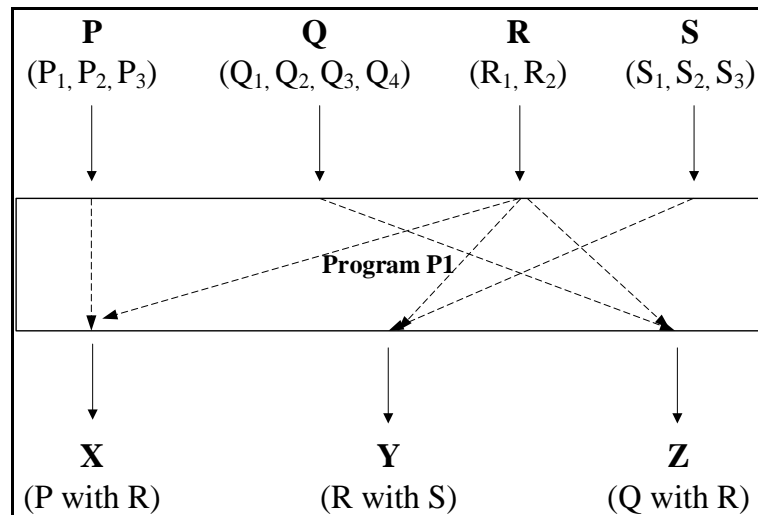


Figure 2.1 Input-output relationship between P, Q, R, S and X, Y, Z of Program P1

Figure 2.1 shows the illustration of the relationship between inputs and outputs used in this example. With IOR, only the combinations of input parameters that will result in corresponding output which are combinations PR, QR and RS will be involved in interaction between the input parameters.

Table 2.4 List of all possible combinations of input values with IOR

PR	QR	RS
P_1R_1	Q_1R_1	R_1S_1
P_1R_2	Q_1R_2	R_1S_2
P_2R_1	Q_2R_1	R_1S_3
P_2R_2	Q_2R_2	R_2S_1
P_3R_1	Q_3R_1	R_2S_2
P_3R_2	Q_3R_2	R_2S_3
	Q_4R_1	
	Q_4R_2	

All combinations of input values that generate the output X, Y and Z are listed in Table 2.4. The total number of possible combinations has reduced from 53 to 20 comparing to uniform interaction strength combinatorial testing where the interaction strength is equal to 2.

Table 2.5 Test suite generated through IOR

Test Case	Input Parameters				Occurrences	Combination Covered
	P	Q	R	S		
1	P ₁	Q ₁	R ₁	S ₁	3	(P ₁ , R ₁), (Q ₁ , R ₁), (R ₁ , S ₁)
2	P ₁	Q ₁	R ₂	S ₁	3	(P ₁ , R ₂), (Q ₁ , R ₂), (R ₂ , S ₁)
3	P ₂	Q ₂	R ₁	S ₂	3	(P ₂ , R ₁), (Q ₂ , R ₁), (R ₁ , S ₂)
4	P ₂	Q ₂	R ₂	S ₂	3	(P ₂ , R ₂), (Q ₂ , R ₂), (R ₂ , S ₂)
5	P ₃	Q ₃	R ₁	S ₃	3	(P ₃ , R ₁), (Q ₃ , R ₁), (R ₁ , S ₃)
6	P ₃	Q ₃	R ₂	S ₃	3	(P ₃ , R ₂), (Q ₃ , R ₂), (R ₂ , S ₃)
7	P ₁	Q ₄	R ₁	S ₁	1	(Q ₄ , R ₁)
8	P ₁	Q ₄	R ₂	S ₁	1	(Q ₄ , R ₂)

Based on the test suite generated through IOR, the complete input-output based relations covering array is IOR (8, 3², 4¹, 2¹, Rel) where Rel = ({P, R}, {Q, R} {R, S}). From Table 2.5, the extent of the test suite produced for all input parameters in Table 2.1 using IOR strategy (12 is much reduced compared to exhaustive testing (72 test cases) and uniform interaction strength combinatorial testing strategy (12 test cases). This happened due to the unwanted combinations of input values have been eliminated from involving in generating test suite. Hence, this prove that IOR approach is efficient enough to trim the amount of test cases generated.

2.2 RELATED WORK

Lately, there are many input-outputs based relation combinatorial testing with different optimization strategies are being studied by researchers. Basically, the optimization strategies used by IOR combinatorial testing can be classified into two types which are pure computational approach and nature based approach (AbdulRahman A Alsewari, Tairan, & Zamli, 2015).

2.2.1 Pure Computational Approach

Pure computational approach is a way that generate the test suite through greedy and iteratively process. The strength of this process is it searches for possible combinations in the search space and terminate upon all combinations are covered. However, the weakness of pure computational approach is too many combinations will cause the approach to become impractical and the cost will increase as well (Xiang, Alsewari, & Zamli, 2015). Moreover, computational using greedy strategies often will

get trapped in local optima (Wu, Nie, Kuo, Leung, & Colbourn, 2015). There are few examples of pure computational approach which are Greedy (Cormen, Leiserson, Rivest, & Stein, 2009), Density (Colbourn, Cohen, & Turban, 2004), TVG (Yu-Wen & Aldiwan, 2000), Union (Patrick J. Schroeder, 2001), ITTDG (Othman & Zamli, 2011), ReqOrder (Ziyuan, Changhai, & Baowen, 2007), ParaOrder (Z. Y. Wang, B. W. Xu, & C. H. Nie, 2008), AURA (Ong & Zamli, 2011), Automatic Efficient Test Generator (AETG) (D. M. Cohen, Dalal, Parelus, Patton, & Bellcore, 1996), IPOG (Lei, Kacker, Kuhn, Okun, & Lawrence, 2007) and Jenny (Jenkins, 2003). Greedy, Density and AURA approaches are selected and their details are discussed at below.

2.2.1.1 Greedy Algorithm

Greedy algorithm is introduced in the book of Introduction to Algorithms (Cormen et al., 2009). Basically, Greedy algorithm always selects the choice that are the best at the moment. Selecting a locally optimal choice is in hope to get the chance that leads to obtain the globally optimum solution. Not every optimization problem can be solved by generating the optimum result using Greedy algorithm, but it works well for most of the problems.

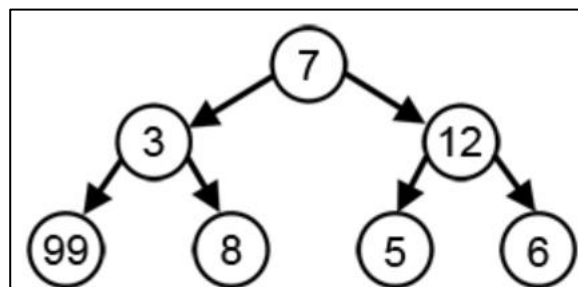


Figure 2.2 Example to show Greedy algorithm's problem

The purpose of this example is to find out the largest path cost of this tree. Based on Figure 2.2, considering the path cost is the number inside the node, the initial node of the tree is node 7 and it has two child nodes which are node 3 and node 12. By using Greedy algorithm, the selection of the next node will be node 12 since the value of node 3 is smaller than node 12. After that, it will continue with the selection of the children of node 12 which are node 5 and node 6. Once again, node 6 will be selected as the value of node 6 is bigger than node 5. So, the total path cost using Greedy algorithm is 25. In this example, the highest path cost should be 109 (the combination of path 7, path 3 and path

99). This shows the Greedy algorithm is not always optimum and it will trap in its local optima and it may not competent to attain the global optimum. Figure 2.3 shows the basic operation of recursive Greedy algorithm.

```

RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]    // find the first activity in Sk to finish
3      m = m + 1
4  if m ≤ n
5      return {am} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅

```

Figure 2.3 Pseudocode of recursive Greedy algorithm (Cormen et al., 2009)

Greedy algorithm has previously used by Schroeder, the researcher who proposed combinatorial testing with IOR feature in one of his study. This research is about the strategy to generate the expected output of a test suite for automated black box testing (P. J. Schroeder, Faherty, & Korel, 2002). This research has applied input-output based relation combinatorial testing to determine particular combinations of inputs which affect the outputs of program. Furthermore, the study of variable strength combinatorial test suite using Greedy algorithm is done by Wang and his colleagues to increase the flexibility of controlling the interaction strength (Z. Wang, B. Xu, & C. Nie, 2008).

2.2.1.2 Density

Density is a concept in optimizing the generation of the best single test case which proposed by Colbourn, Cohen and Turban in their research in pairwise testing (Colbourn et al., 2004). There are two types of density which are local density and global density. Global density is used in test case generation and it is calculated based on local density. For each coverage requirement r_k ($1 \leq k \leq t$), local density can be defined as below.

$$LD_k = \frac{num_k}{(\max_{1 \leq k \leq t} \{\prod_{f_i \in r_k} a_i\})^{(n_k - p_k)/n_k}}$$

num_k represents the number of combinations that have not been covered yet in set $CombSet_k$ where the values of such aspects are equivalence to rigid values in present test case. The value of p_k is the fixed number of aspects. The maximum available combination is one if the value of p_k is equal to n_k . The density is equal to 1 whenever there is any

combination that covers an uncovered combination else the density is 0. The larger the value of local density of each coverage requirement, the more the combinations covered by the test case. Deriving from the definition above, the global density can be represented as below.

$$GD = \sum_{k=1}^t LD_k$$

The global density should be high to ensure the test case generated are capable to cover uncover combinations as much as possible when generates a test case. Figure 2.4 shows the pseudocode of generating test case using Density concept.

```

Algorithm 2. Generate Single Test Case
Start with an empty test case test
While (at least one coverage requirement has not
been handled)
  For  $k=1$  to  $t$ 
    If (in  $r_k$ , at least one factor whose value has
not been fixed) then
      Calculate local density for  $r_k \in R$ 
    End If
  End For
  Select a new coverage requirement  $r_k$  with the
greatest local density
  For each  $comb \in CombSet_k$ 
    If ( $comb$  is available in test) then
      Calculate global density by assuming the
values of factors in  $r_k$  are fixed as  $comb$ 
    End If
  End For
  Select a combination  $comb$  that takes the greatest
global density
  Fix factors in  $r_k$  as the selected combination
End While

```

Figure 2.4 Pseudocode of test case generation using concept of Density (Z. Wang et al., 2008)

Density based algorithm was first applied in the research in interaction testing by Colbourn and his associates but it was limited only to pairwise testing (Colbourn et al., 2004). Colbourn and his colleagues are then continue the research by increasing the limit of interaction strength (R. C. Bryce & Colbourn, 2009). There is another research conducted by Wang and his colleagues that introduced density based algorithm in variable strength interaction combinatorial testing (Z. Wang et al., 2008).

2.2.1.3 AURA

AURA is a non-deterministic input-output based relationship combinatorial testing strategy that proposed by Ong and Kamal (Ong & Zamli, 2011). This strategy is focusing in solving the mapping of symbolic values to actual data manually and the lack of flexibility of existing test suite generation. Automated input-output mapping is implemented to reduce the time and cost consumed as well as the mistake that may made by software testers when they are carrying out the mapping process. Besides, AURA has managed to increase the optimality of the size of test suite and decrease the time taken for the test suite generation. Figure 2.5 is the overview of AURA strategy in generating the test suite for combinatorial testing.

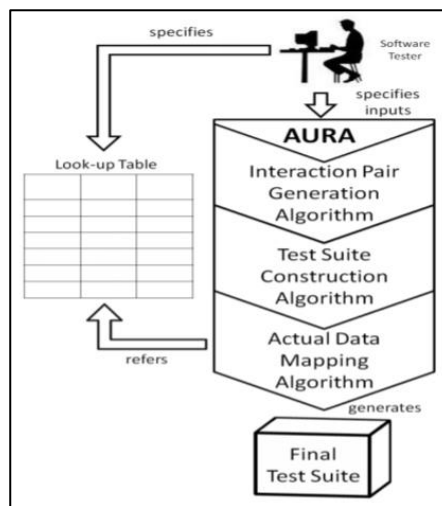


Figure 2.5 Overview of AURA strategy (Ong & Zamli, 2011)

AURA strategy starts with software tester specifies the input in actual data form into the look-up table for system under test as well as designates the symbolic values of inputs into AURA strategy. AURA strategy is built based on three algorithm which are interaction pair generation algorithm, test suite construction algorithm and actual data mapping algorithm. The first algorithm of AURA strategy is triggered once the symbolic values are entered in the AURA strategy to bring about all potential interaction pairs. All generated interaction pairs are then used to form the test suite using test suite construction algorithm. Lastly, actual data mapping algorithm is applied to come out with the final test suite that is in the actual data form based on the predefined look-up table. Figure 2.6, 2.7 and 2.8 shows the pseudocode for interaction pair generation algorithm, test suite construction algorithm and actual data mapping algorithm respectively.

```

Algorithm of Interaction Pair Generation(factor set, F and interaction coverage requirements, R)
1: begin
2: let  $F = \{f_0, \dots, f_m\}$  where F represents number of values defined for each factor, f and m =
   number of factor
3: let  $R = \{r_0, \dots, r_j\}$  where R represent the set of interaction coverage requirements
4: initialize  $IE = \{\}$ , where IE represents interaction pairs partitioned base data structure
5: for each interaction coverage requirement,  $r_i$ 
6: {
7:   for each combination of parameter interaction in  $r_i$ 
8:   {
9:     if (factors are interaction elements)
10:      generate interaction pair for interaction elements;
11:     else
12:      append don't care ("X") values for non-interacting elements;
13:      store the interaction pair (with "X") into  $IE[i]$ ;
14:   }
15: }
16: return IE;
17: end

```

Figure 2.6 Pseudocode of interaction pair generation algorithm (Ong & Zamli, 2011)

```

Algorithm of Test Suite Construction (number of iterations, n and interaction pairs, IE)
1: begin
2: initialize  $\tau = \{\}$ , where  $\tau$  represents final test suite
3: initialize  $a = \{\}$ , where a represents a test case that will be comprised of a set of
   factors
4: initialize  $\beta = \{\}$ , where  $\beta$  represents covered pairs corresponding to a
5: for each interaction coverage requirement,  $r_i$ 
6: {
7:   for each combination of parameter interaction in  $r_i$ 
8:   {
9:     generate interaction pair for interaction elements;
10:    for n times of iterations
11:    {
12:      assign a random value for each non-interacting element;
13:      form a;
14:      check  $\beta$  on IE;
15:      if (maximum # of  $\beta$ )
16:        break;
17:      else
18:        update weight;
19:    }
20:    add a into  $\tau$ ;
21:    remove  $\beta$  in IE;
22:    if (IE is null)
23:      break;
24:  }
25: if (IE is null)
26:   break;
27: }
28: return  $\tau$ 
28: end

```

Figure 2.7 Pseudocode of test suite generation algorithm (Ong & Zamli, 2011)


```

Algorithm of Actual Data Mapping ( $\tau$ , output file name)
1: begin
2: define ofile as output file name
3: initialize  $\sigma = \{\}$  where  $\sigma$  represents the list of actual data
4: if (look-up table file is specified)
5: {
6:   translate actual data from look-up table file to  $\sigma$ ;
7:   for each test case in  $\tau$ 
8:   {
9:     convert test case into actual data based on  $\sigma$ ;
10:    write test case into ofile;
11:   }
12: }
13: else
14: {
15:   for each test case in  $\tau$ 
16:     {write test case into ofile;}
17: }
18: end

```

Figure 2.8 Pseudocode of actual data mapping algorithm (Ong & Zamli, 2011)

2.2.1.4 Comparison between pure computational approaches

The maximum interaction strength supported and the presence of input-output relation feature by each approach are recorded in Table 2.6.

Table 2.6 Comparison between natural based approaches

Approaches	Maximum interaction strength support	IOR Support
Greedy algorithm (Z. Wang et al., 2008)	3	Yes
Density algorithm (R. Bryce & Colbourn, 2007)	4	Yes
AURA (Ong & Zamli, 2011)	3	Yes

2.2.2 Natural Based Approach

The nature based approach is inspired by the behaviour of natural like such as ant colony, the gene of the chromosome, swarm of birds and so on. There are many approaches such as Ant Colony Optimization Algorithm (ACO) (Blum, 2005), Genetic algorithm (GA) (Holland, 1992), Harmony Search algorithm (HS) (Z. W. Geem & Kim,

2001), Simulated Annealing (SA) (M. B. Cohen, Gibbons, Mugridge, & Colbourn, 2003) and Particle Swarm Optimization (PSO) (Eberhart & Kennedy, 1995) that are being implemented in combinatorial testing but only one of them supports IOR feature which is Ant Colony Optimization algorithm (Ramli et al., 2016). The details of ACO are explained as below.

2.2.2.1 Ant Colony Optimization (ACO)

Ant Colony optimization algorithm is a strategy which first introduced by Marco Dorigo and his associates (Blum, 2005). Basically, ACO is inspired by the way on how a colony of ants seeks for the shortest pathway when they are finding the food sources from their nest. At first, ants will seek the surrounding in the random manner and they will leave a trail of chemical pheromone on the ground while moving. Ants will then determine the road path that with strong pheromone by smelling it. After the ants found the food source, they will analyse the quality and quantity of the food and carry any of the food they afford. The ants will leave the pheromone along the road path they go back to their nest and the concentration of pheromone leave is decided based on quantity and quality of the food. Other ants will follow the pheromone that leave by the previous ant to carry the food back to their nest. Once the food is fully collected, the ants will not continue to leave the pheromone on the same track anymore instead they will start exploring the new path to get new food. The pheromone on the track will eventually evaporate. This phenomena is known as stigmergy which favours the ants to discover the shortest pathway between food source and their nest by communicating indirectly through pheromone (Blum, 2005). Figure 2.9 shows the algorithm of ACO while the process of finding the shortest path from their nest to the food source using pheromone is illustrated in Figure 2.10.

```

Algorithm 1. Ant colony optimization (ACO)
while termination conditions not met do
    ScheduleActivities
        AntBasedSolutionConstruction() {see Algorithm 2}
        PheromoneUpdate()
        DaemonActions() {optional}
    end ScheduleActivities
end while

Algorithm 2. Procedure AntBasedSolutionConstruction() of Algorithm 1
s = <>
Determine N (s)
while N (s) ≠ ∅ do
    c ← ChooseFrom(N (s))
    s ← extend s by appending solution component c
    Determine N (s)
end while

```

Figure 2.9 Pseudocode of Ant Colony Optimization (Blum, 2005)

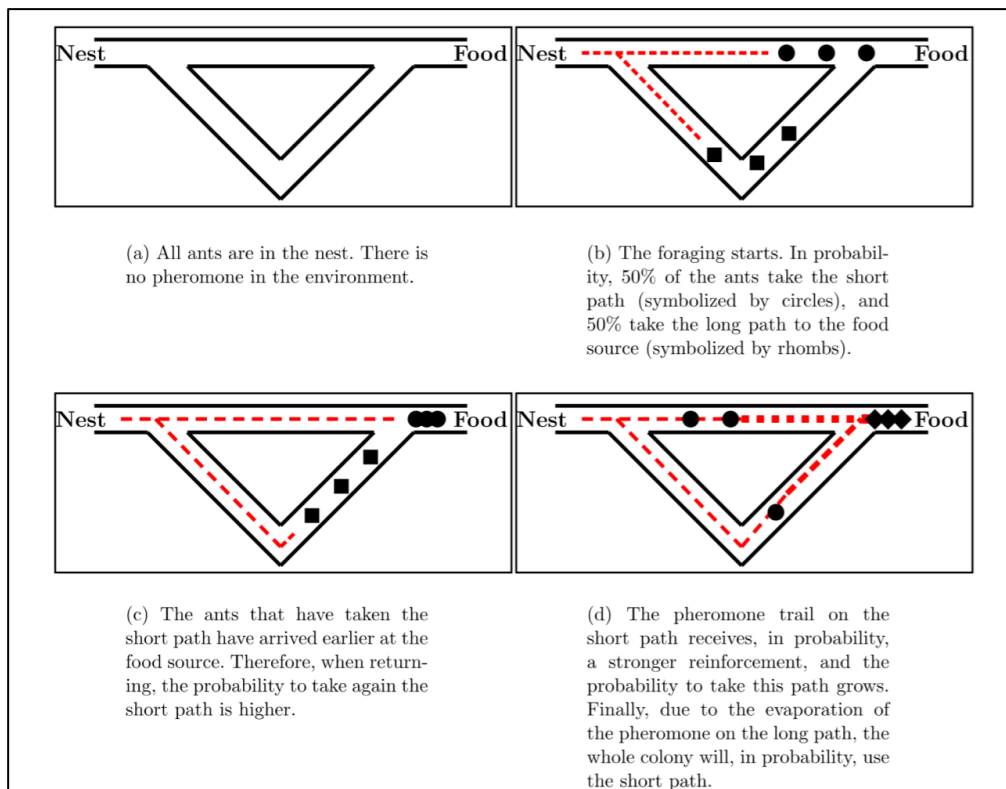


Figure 2.10 The illustration on how ants find the shortest path between food source and their nest (Blum, 2005)

There are few advantages of Ant Colony Optimization which are the speed in finding good solutions and inherit parallelism (Selvi & Umarani, 2010). Moreover, ACO is capable to select the best edge at the very beginning of the execution of this algorithm (Ramli et al., 2016). Furthermore, ACO also able to avoid premature convergence as well

as mutually explore the search space (Liang et al., 2014). Conversely, uncertain time for convergence is one of the disadvantages of ACO. The probability distribution varies by iteration and the research is more towards experimental instead of theoretical. In addition, the sequences of random decisions are dependent as well as the theoretical analysis is complex (Selvi & Umarani, 2010).

There are several researches have been done on showing the implementation of Ant Colony Optimization algorithm in combinatorial testing. These include the study of comparing the efficiency of ACO with simulated annealing and genetic algorithm (Mao, Yu, Chen, & Chen, 2012) as well as applying ACO in variable interaction strength combinatorial testing (X. Chen, Gu, Li, & Chen, 2009). ACO has been proposed to be applied in IOR combinatorial testing in the recent research by Ramli and her associates (Ramli et al., 2016) but there is lack of result of implementation in the study.

2.2.2.2 Genetic Algorithm

Genetic algorithm (GA) is one of the metaheuristic algorithms that belongs to evolutionary algorithm that most often used in solving combinatorial optimization problem. GA was proposed by John Holland back in 1992 (Holland, 1992). This algorithm possesses the same principle as Darwin's principle of evolution by applying the selection of gene in each generation. Initially, a population of chromosomes is randomly generated. Then, the process of selecting the chromosomes based on the fitness function is happened and the chromosomes are then recombined by crossover their genes between the pair of chromosomes to produce offspring. The next generation of population are born after the combination of chromosomes. The iteration of this process is then continued and the successive generation of chromosomes are evolved to become better with the improved fitness function. The iteration terminates when the stopping criteria is achieved (McCall, 2005). Figure 2.11 shows how the crossover process is happened while Figure 2.12 is the pseudocode for GA.

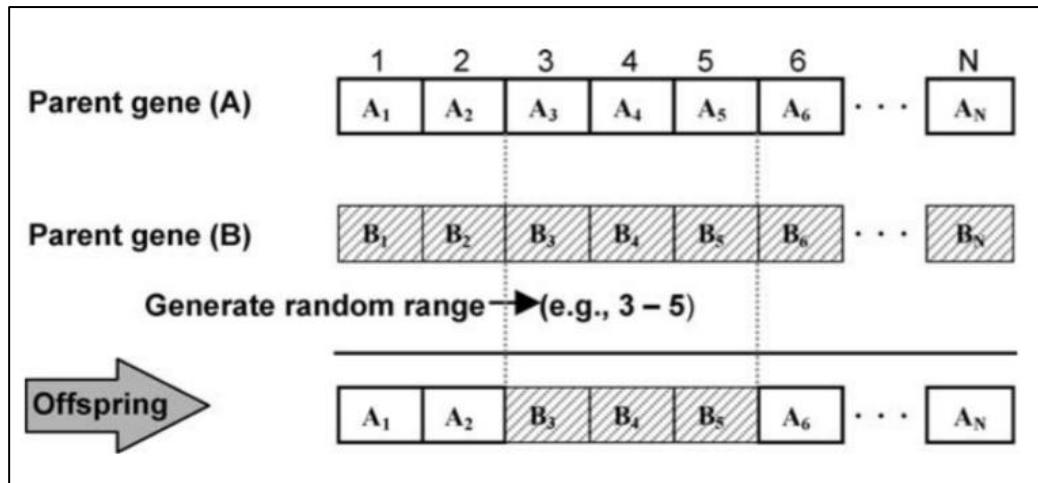


Figure 2.11 Crossover operation to generate offspring (Elbeltagi, Hegazy, & Grierson, 2005)

```

Begin;
  Generate random population of  $P$  solutions
  (chromosomes);
  For each individual  $i \in P$ : calculate fitness ( $i$ );
  For  $i = 1$  to number of generations;
    Randomly select an operation (crossover or
    mutation);
    If crossover;
      Select two parents at random  $i_a$  and  $i_b$ ;
      Generate on offspring  $i_c = \text{crossover}(i_a \text{ and } i_b)$ ;
    Else If mutation;
      Select one chromosome  $i$  at random;
      Generate an offspring  $i_c = \text{mutate}(i)$ ;
    End if;
    Calculate the fitness of the offspring  $i_c$ ;
    If  $i_c$  is better than the worst chromosome then
    replace the worst chromosome by  $i_c$ ;
  Next  $i$ ;
  Check if termination = true;
End;

```

Figure 2.12 Pseudocode for Genetic Algorithm (Elbeltagi et al., 2005)

GA is well adapt in the optimization problems such as the structure of the search space is irregular and the situation where the search is computationally intractable (McCall, 2005). Besides, GA has the ability to extract rules which are considered simple to understand (Shin & Lee, 2002). However, GA has some disadvantages and the most significant one is the speed of finding the solution using GA is very slow (Abramson &

Abela, 1991). In addition, GA tends to get trapped in local optima and the speed convergence is slow as well as it contain non-explicit memorization of best individuals (Lam, Raju, M, Ch, & Srivastav, 2012).

Generally, GA has been implemented in solving combinatorial testing optimization problem such as in the research by Shiba and his associates (Shiba, Tsuchiya, & Kikuno, 2004a). Srivastava and Kim developed variable strength interaction combinatorial testing using GA to focus on the parts that are critical by implementing a more selective approach (Srivastava & Kim, 2009). Furthermore, McCaffrey conducted a study to identify the effectiveness of GA in pairwise testing (McCaffrey, 2009). Nevertheless, GA has not yet been applied for the optimization in combinatorial testing that has IOR feature.

2.2.2.3 Harmony Search Algorithm

Harmony Search (HS) algorithm is a new metaheuristic optimization algorithm introduced by (Zong Woo Geem, Kim, & Loganathan, 2001) back in 2001. HS algorithm is inspired by the spontaneity of playing music by an experienced musician. There are three ways to play a music spontaneously by a skilled musician. The first way is playing any well-known melody literally from his mind. Playing with the aforementioned melody with marginally adjusted pitch is the second way while the third way is playing some random or completely new notes. The ways of playing music instinctively have been defined into quantitative optimization process by (Zong Woo Geem et al., 2001) which consists of three main components which are harmony memory (HM), pitch adjustment and randomization. HS algorithm always search for global optimum based on objective function by going through iteration.

Harmony Search algorithm starts with initialization of Harmony Memory. The size of HM is determined by harmony memory size (HMS). HM plays role in containing the candidates that sorted by best objective values. Next, a new Harmony is improvised based on the value of Harmony Memory Considering Rate (HMCR) and Pitch Adjustment Rate (PAR) to enhance the HM. Moreover, replacement of minimum harmony by new harmony generated in HM takes place when new harmony is better than the minimum harmony in HM. The process undergoes iteration until the stopping criteria is achieved. The pseudocode of HS algorithm is shown in Figure 2.13.

```

Begin
Define objective function  $f(\mathbf{x})$ ,  $\mathbf{x}=(x_1, x_2, \dots, x_n)^T$ 
Define harmony memory accepting rate ( $R_{HMCR}$ )
Define pitch adjusting rate ( $R_{PAR}$ ) and other parameters
Generate Harmony Memory with random harmonies
While ( $t < \text{max number of iterations}$ )
  While ( $i \leq \text{number of variables}$ )
    If ( $R_{\text{random}} \leq R_{HMCR}$ ), Choose a value from HM for the variable  $i$ 
    If ( $P_{\text{random}} \leq P_{PAR}$ ), Adjust the value by moving to next or previous value
    Else Do not adjust the value chosen from HM
    Else Choose a random value
  End while
  Accept and add the New Harmony (solution) to HM if better than the worst
  harmony
End while

```

Figure 2.13 Pseudocode of Harmony Search algorithm (Abdul Rahman, 2012)

HS algorithm possess a few advantages compared to traditional methods. Firstly, HS algorithm used simple mathematical operations and randomly selects the control variables as well as the search process of HS algorithm is run randomly (Khazali & Kalantar, 2011). However, HS algorithm also has some limitations. Notably, the decision variables of other harmony vectors that keep in HM often being selected to become the decision variables of a new harmony. Moreover, taking a place by new harmony vector in the memory may happens after the fitness test of new harmony vector. These issues cause the time taken for HS to be converged to the global optimum is affected (Ammar, Bouaziz, Alimi, & Abraham, 2013).

There are several researches have been applied Harmony Search algorithm in combinatorial testing. In 2011, Alsewari and his associates are the first to apply HS algorithm in t-way interaction test data generation (Abdulrahmn A. Alsewari & Zamli, 2011). Besides, HS algorithm is being implemented in pairwise testing strategy (PHSS) and PHSS is outperformed existing strategies in term of the size of test suite generated in the study in (Abdul Rahman, 2012). In addition, utilization of HS algorithm in variable strength interaction combinatorial testing with constraint support is carried out by (A. R. A. Alsewari & Zamli, 2012). There is one research regarding t-way testing using HS algorithm that supports input-output relation feature. However, this research is not yet published.

2.2.2.4 Particle Swarm Optimization

Particle Swarm Optimization (PSO) algorithm was introduced by James Kennedy and Russell Eberhart back in 1995 (Kennedy & Eberhart, 1995). PSO is a population based stochastic algorithm in solving optimization problem. This algorithm was inspired by the act of a flock of birds and a school of fish where they possess the swarming behaviour in nature. PSO uses particles to make up its own population which will move in n-dimensional, real-valued search space to get the possible solution for the problem faced (Vesterstrom & Thomsen, 2004). Each particle has three D-dimensional vectors which are current position, previous best position as well as the velocity. The current position represents a problem solution and it will be stored as previous best position if it is better than any previously found solutions. Velocity is used to determine the step size and failure to tune the velocity will affects the performance of the algorithm (Poli, Kennedy, & Blackwell, 2007). The pseudocode of PSO is shown as below.

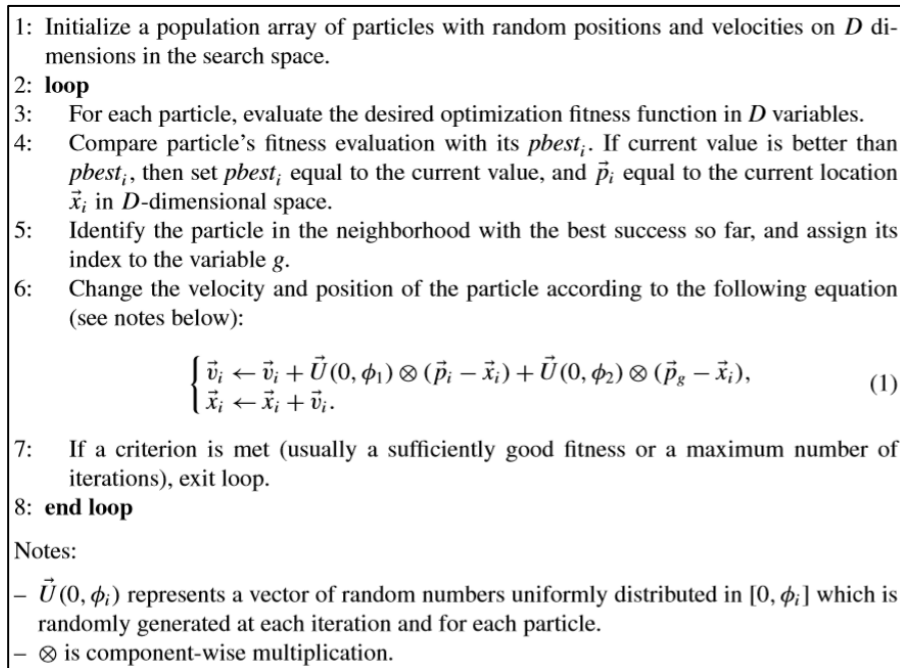


Figure 2.14 Pseudocode of Particle Swarm Optimization (Poli et al., 2007)

Based on the findings, PSO have been applied in pairwise testing by China researchers in 2010 (Xiang Chen, Gu, Qi, & Chen, 2010). Other than that, test suite generation using variable interaction strength also implemented PSO to solve the combinatorial problem (Ahmed & Zamli, 2011). Still, there is no combinatorial testing employ IOR feature using PSO.

2.2.2.5 Simulated Annealing

Simulated Annealing (SA) is a metaheuristic algorithm which used probabilistic techniques to estimate the global optimum of a problem. SA is inspired by the effect of slow cooling process on the molecule of a metallic substance. The cooling of molecules will make the molecules slowly converge toward optimal rest energy and SA replicates this phenomena in the algorithm (Stardom, 2001). SA can optimize process cost functions which has random degree of nonlinearities, discontinuities and stochasticity and it is proven to have an optimal solution when executed. SA is considered easy to be implemented compared to other nonlinear optimization algorithms in term of coding (Ingber, 1993). The concept of SA is similar to hill climbing but it has additional feature which is probability can be controlled. This feature is to control whether to reduce the quality of current solution which will prevent from getting stuck in a bad configuration in the searching process is going on (M. B. Cohen, Gibbons, et al., 2003). The details on how SA works is shown in the pseudocode below.

```
Select an initial temperature  $T_0 > 0$ ;  
Select an initial solution,  $S_0$ , and make it the current solution,  $S$ , and the current best solution,  $S^*$ ;  
repeat  
  set repetition counter  $n = 1$ ;  
  repeat  
    generates solution  $S_n$  in the neighbourhood of  $S$ ;  
    calculate  $\Delta = f(S_n) - f(S)$ ;  
    if ( $\Delta \leq 0$ ) then  $S = S_n$ ;  
    else  $S = S_n$  with the probability of  $p = e^{-\Delta/T}$ ;  
    if ( $f(S_n) < f(S^*)$ ) then  $S^* = S_n$ ;  
     $n = n + 1$ ;  
  until  $n >$  number of repetitions allowed at each temperature level ( $L$ );  
  reduce the temperature  $T$ ;  
until stop criterion is true.
```

Figure 2.15 Pseudocode of Simulated Annealing algorithm (Xambre & Vilarinho, 2003)

Furthermore, there are few researches that related to combinatorial testing using SA for optimization have been carried out. Cohen and Colbourn used SA to solve the optimization problem while constructing test suite for interaction testing (M. B. Cohen, Gibbons, et al., 2003). SA also used to combine with algebraic construction to build covering arrays for interaction testing that is strength three (M. B. Cohen, Colbourn, & Ling, 2003). Again, there is no research which used SA algorithm to optimize the combinatorial problem in combinatorial testing that features IOR.

2.2.2.6 Comparison between natural based approaches

Table 2.7 presents the maximum interaction strength support and the availability of input-output relation of each natural based strategy.

Table 2.7 Comparison between natural based approaches

Approaches	Maximum interaction strength support	IOR Support
Ant Colony Optimization (Shiba et al., 2004a)	3	P/S
Genetic algorithm (Shiba et al., 2004a)	3	No
Harmony Search algorithm (Abdulrahmn A. Alsewari & Zamli, 2011)	6	N/P
Particle Swarm Optimization (Ahmed, Zamli, & Lim, 2012)	6	No
Simulated Annealing (M. B. Cohen, Gibbons, et al., 2003)	3	No

Note:

P/S represents only strategy is proposed

N/P represents not published

2.3 JAYA ALGORITHM

Swarm Intelligence (SI) is one of the pillars under artificial intelligence (AI) discipline and it became much more well-known over the last decade (Blum & Li, 2008). The concept of SI depends on the mannerism of social swarm of insects and animals like ants, birds, bees and so on. The examples of SI optimization method are ant colony optimization, artificial bee colony and particle swarm optimization. Furthermore, Evolutionary algorithm (EA) which belongs to another AI discipline that proposed to discover the near-optimal solutions for the problem faced. The most notably EA algorithms are Evolution Programming, Genetic Algorithm and Evolution Strategy (Vesterstrom & Thomsen, 2004).

Both swarm intelligence and evolutionary algorithms are probabilistic algorithm which need common controlling parameters such as number of generations and population size as well as each algorithm-specific control parameters (R Rao, 2016). For instance, Genetic Algorithm uses selection operator, crossover probability and mutation probability as its own algorithm-specific control parameters. Every parameter in an

algorithm must be tuned properly including algorithm-specific control parameters to ensure the algorithm can perform well. Failure of tuning the algorithm-specific control parameters will cause the solution being trapped in local optimum or long computational time needed.

To conquer the necessity of tuning of algorithm-specific control parameters, teaching-learning-based optimization (TLBO) algorithm is proposed by Rao and his associates (R. V. Rao, Savsani, & Vakharia, 2011). The advantage of this algorithm is no algorithm-specific control parameter is needed and it needs only common controlling parameters which are number of generations and population size to work. However, TLBO algorithm needs two phases which are teacher and learner phase to function. Therefore, Rao and his colleagues introduced a simpler algorithm that has only one phase which is Jaya algorithm (R Rao, 2016).

Jaya algorithm begins with the initialization of population size, number of design variables, termination criterion and a population of solutions by an objective function. Next, the iteration of improvement begins with identifying the best and worst solution in the population based on the function value. After that, the value of each design variable will be modified using the formula that proposed in Jaya algorithm. At the end of the iteration, the modified solution will compare with original solution using function value to determine if the modified solution has improved. If the modified solution is better than the original solution, the original solution will be replaced by modified solution. The next iteration will start with the new population of solutions that are improved using modified solutions. The iteration continues until the termination criterion which is number of iterations for improvement is achieved. Below shows the flowchart of Jaya Algorithm.

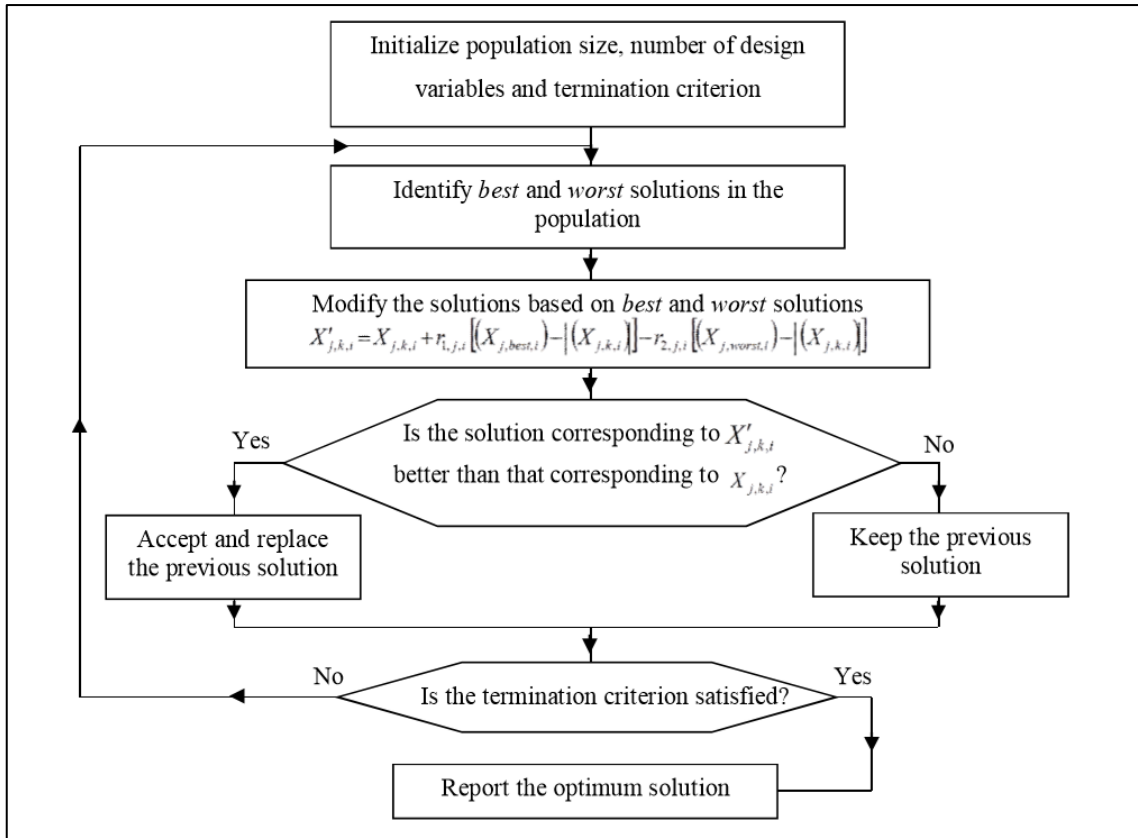


Figure 2.16 Flowchart of Jaya Algorithm (R Rao, 2016)

The main purpose of Jaya algorithm is to find the minimum or maximum solution of an objective function, $f(x)$. There are three variables that are needed in this algorithm and they are stated as following:

- a. Number of iteration (i): The total rounds of improvements
- b. Number of design variables (m): The number of parameters in objective function, $j = 1, 2, \dots, m$
- c. Number of candidate solutions (n): The size of population, $k = 1, 2, \dots, n$

Assume that the best candidate solution (*best*) has the best value of $f(x)$ while the worst candidate solution (*worst*) has the worst value of $f(x)$ in the entire candidate solutions and $X_{j,k,i}$ is the value of the j^{th} design variable for k^{th} candidate solution in i^{th} iteration, then the following equation is fulfilled aforementioned requirements.

$$X'_{j,k,i} = X_{j,k,i} + r_{1,j,i} (X_{j,best,i} - |X_{j,k,i}|) - r_{2,j,i} (X_{j,worst,i} - |X_{j,k,i}|)$$

$r_{1,j,i}$ and $r_{2,j,i}$ are two randomly generated number for j^{th} design variable in i^{th} iteration which is in the range of 0 to 1. These random numbers assure the search space is explored well (RV Rao, More, Taler, & Ocloń, 2016). $X_{j,best,i}$ and $X_{j,worst,i}$ serve as the value of the design variable j for the *best* and *worst* candidates respectively. The presence of $|X_{j,k,i}|$ is to make sure the exploration could be done even better (RV Rao et al., 2016). The tendency of the solution to move closer to the best solution is expressed by $r_{1,j,i} (X_{j,best,i} - |X_{j,k,i}|)$ while $r_{2,j,i} (X_{j,worst,i} - |X_{j,k,i}|)$ denotes the tendency of the solution to stay away from worst solution. $X'_{j,k,i}$ represents the updated value of $X_{j,k,i}$ and it will only be accepted if it delivers better function value.

Jaya Algorithm has several advantages over other optimization algorithm as affirmed by the researchers. Firstly, Jaya algorithm are free from algorithm-specific parameters which result in less computational effort is needed and the complexity of the algorithm is decreased (Singh, Prakash, Singh, & Babu, 2017). Moreover, Jaya algorithm has the ability to avoid the solution from trapping in local optima compare to other optimization algorithm (Warid, Hizam, Mariun, & Abdul-Wahab, 2016). Additionally, the ease of resolving discrete optimization problem and convergence to global optimum value make Jaya algorithm even better than other optimization algorithms (Mishra & Ray, 2016). Furthermore, Jaya algorithm is found out that the speed of convergence to reach the global solution is faster than TLBO algorithm which indicates the time to compute will be much shorter (Mishra & Ray, 2016).

Jaya Algorithm is considered as relatively new optimization algorithm as it was just introduced back in 2016. However, there are a number of researches that adopted Jaya algorithm in solving the optimization problems. Dimensional optimization of a micro-channel heat sink is optimized using Jaya algorithm (RV Rao et al., 2016). Besides, Jaya algorithm also applied in optimizing the coefficients of proportional plus integral controller and filter parameters of photovoltaic fed distributed static compensator (PV-DSTAT-COM) (Mishra & Ray, 2016). In addition, Jaya algorithm also used to minimize the single objective of performance measure of a proportional-integral-derivative (PID) controller for automatic generation control (AGC) of an interconnected power system (Singh et al., 2017).

Table 2.8 Comparison between Jaya algorithm with existing strategies

Approaches	Maximum interaction strength support	IOR Support
Greedy algorithm (Z. Wang et al., 2008)	3	Yes
Density algorithm (R. C. Bryce & Colbourn, 2009)	3	Yes
AURA (Ong & Zamli, 2011)	3	Yes
Ant Colony Optimization (Shiba et al., 2004a)	3	P/S
Genetic algorithm (Shiba et al., 2004a)	3	No
Harmony Search algorithm (Abdulrahmn A. Alsewari & Zamli, 2011)	6	N/P
Particle Swarm Optimization (Ahmed et al., 2012)	6	No
Simulated Annealing (M. B. Cohen, Gibbons, et al., 2003)	3	No
Jaya algorithm	4	Yes

Note:

P/S represents only strategy is proposed

N/P represents not published

From Table 2.8, although there are so many researches have been done to overcome combinatorial optimization problem but until now there has nobody implemented Jaya Algorithm in combinatorial testing for both uniform strength interaction and IOR. Furthermore, there is lack of population-based strategies which is fast in getting the solution have applied in combinatorial testing. Hence, this thesis is going to propose the implementation of Jaya algorithm in input-output based relation combinatorial testing.

CHAPTER 3

METHODOLOGY

3.1 INTRODUCTION

This chapter discusses about the process of input-output based test case generation using Jaya algorithm. The data set that are going test the efficiency of Jaya algorithm is based on the previous study published by (AbdulRahman A Alsewari et al., 2015). The discussion begins with the methodology implemented in this study, hardware and software requirement, Gantt chart and testing plan for the evaluation of the competence of Jaya algorithm in input-output based combinatorial testing.

3.2 METHODOLOGY

Research methodology which comprised of five vital phases is employed in this study. The five essential phases in research methodology start with conduct literature review on existing study, design the solution for IOR combinatorial testing using Jaya algorithm, carry out the implementation of the solution designed, test and evaluate the performance of Jaya algorithm based IOR combinatorial testing, followed by documenting all of the findings as the last step. The overall flow of the methodology is shown as in Figure 3.1.

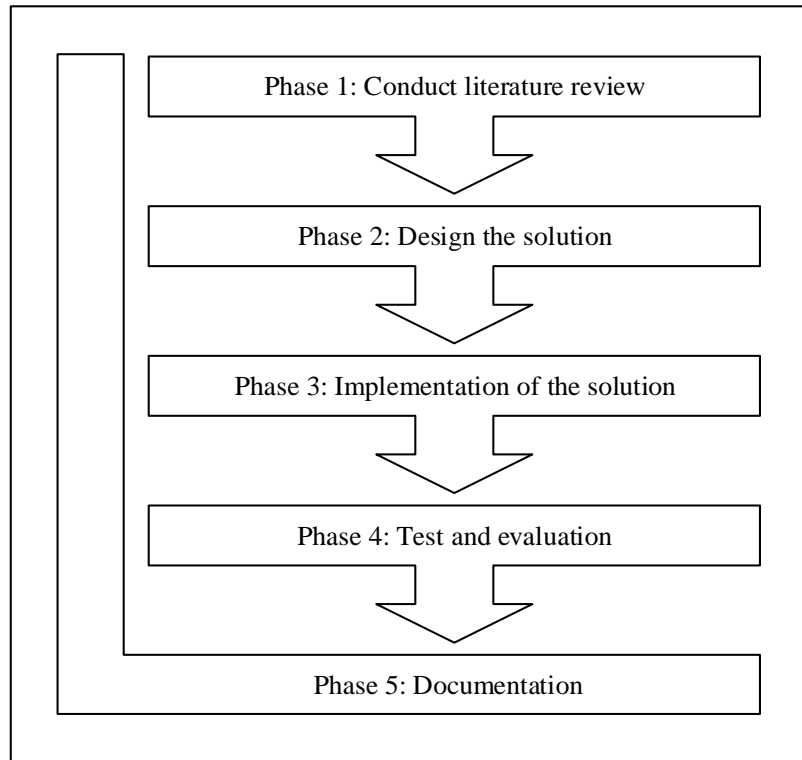


Figure 3.1 The flowchart of research methodology

3.2.1 Phase 1: Literature Review

At the very beginning of conducting a research, the exploration did by other researchers have to be studied first in order to summarize the facts of interested topic. The literature review starts with the background of combinatorial testing including uniform interaction strength, variable interaction strength and input-output relation. The background study is to understand what combinatorial testing is, its types and the difference between uniform interaction strength and input-output based relation combinatorial testing. Later, the review of existing combinatorial testing that applied different optimization strategies is carried out. Based on the review, there are two types of approaches in solving combinatorial optimization problems which are pure computational and natural based approaches. Furthermore, Jaya algorithm, one of the population-based optimization algorithms, is being studied in detail including the algorithm of Jaya, advantages and disadvantages of Jaya algorithm as well as the application of Jaya algorithm in solving real world problems. From the revision of current optimization algorithms, there is still a gap for improvement in combinatorial testing. Hence, Jaya algorithm is proposed to increase the effectiveness of combinatorial optimization in software testing.

3.2.2 Phase 2: Design the Solution

To realize the input-output based relation combinatorial testing based on Jaya algorithm, there are five level of actions to be carried out. The five level of actions start with reading of the input values entered by user, data analyzation and data mapping, input values combination generation, test case generation and final test suite generation. The flow of each steps is illustrated in Figure 3.2.

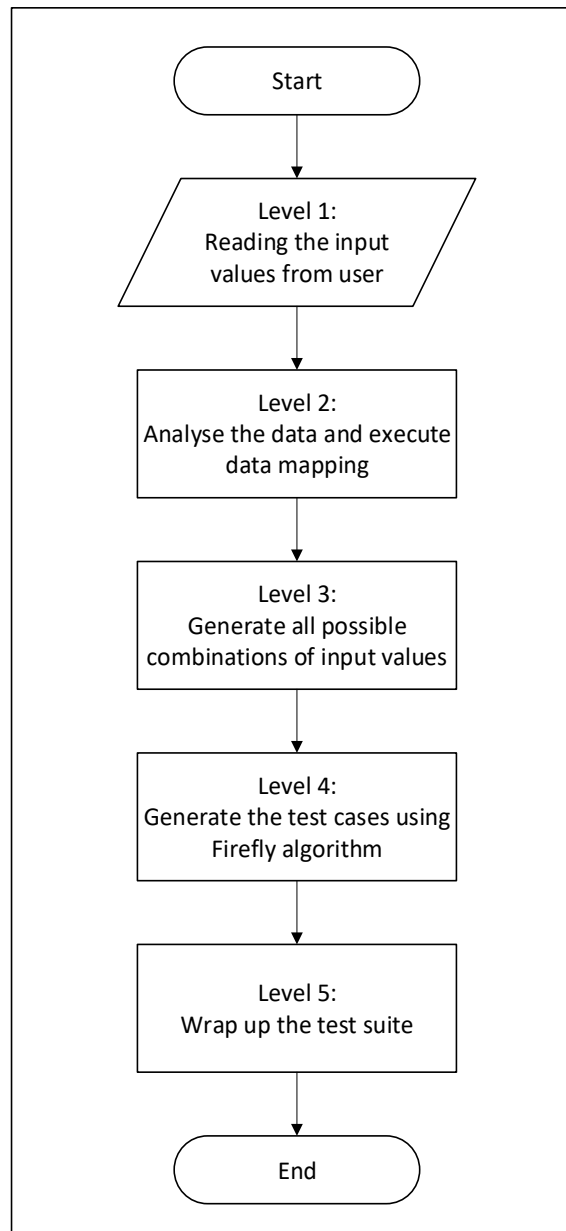


Figure 3.2 The flowchart of the execution of IOR combinatorial testing based on Jaya algorithm

3.2.2.1 Level 1: Reading of input values

The very first step of using IOR combinatorial testing based on Jaya algorithm is read the input values from text box provided where the inputs include number of parameters, number of values inside each parameter, parameter name, name of the values, interaction strength and input-output relationship. After reading the input values, the process is advanced to next level.

3.2.2.2 Level 2: Data analysis and data mapping

After getting the input values, all information entered by the user will be analysed thoroughly. The purpose of analysis is to ensure the information user keyed in is in right format and syntax. Any wrong information inclusive of amiss format and syntax entered will caused system not able to recognize even more the system will crash. Therefore, a preventive action is taken to counter the happening of above situation by giving users feedback message so that they can recheck the problem and make the correction.

Data mapping is carried out right after data analysis. The input values from each parameter that has been verified during data analysis process will undergo mapping process with integers. Take the example from Table 2.1, all input values for each parameter will map with integers starting from 0. The outcome of data mapping is presented in Table 3.1.

Table 3.1 Result of data mapping using the input values from Table 2.1

Parameters	P	Q	R	S
Input values	0	3	7	9
	1	4	8	10
	2	5		11
		6		

By applying data mapping, the time taken to generate all possible combinations of input values as well as the test case will be reduced due to the size of the input data is decreased. The size of a string is larger than an integer in normal case. Smaller bytes of data always process faster than the larger one. Hence, the string values of input data are being substituted with integers when the processing of data takes place.

3.2.2.3 Level 3: Combinations of input values generation

In level 3, the combination of input values is generated to be used in test case generation. Each input value that belongs to the same parameter and have mapped to corresponding integer is merged with other parameters' values to form combinations of values. There are two types of combinations implemented in this project which are combinations of input values based on input-output relationship and interaction strength. The details of IOR and uniform interaction strength combinations is discussed in section 3.2.3.3. Table 2.2 is the example of the combinations of input values between each parameter that are going to be adopted in generating the test case using Jaya algorithm.

3.2.2.4 Level 4: Test case generation based on Jaya algorithm

After gathering all combinations of input values, the next step is to generate the test case. It starts with generating a test case by randomly pick one of the input values from each parameter. The generated test case which will be assessed by determining the number of combinations of input values that generated in step 3 covered by the test case. The best and worst test cases in term of coverage in the population will be picked for modification purpose. Each test case in the population will be improved by applying modification based on the best and worst test cases. If the test case generated after employing the modification has better coverage than the previous one, it will then replace the former test case. After one iteration, the best and worst test cases will be reselected and the modification is done based on the new best and worst test cases. The process is iterated until the maximum number of generations is achieved. The best test case which generated at the end of the iterative process is added into a temporary test suite. The whole process is keep repeating until all combinations of input values are fully covered. These series of actions are illustrated as shown in Figure 3.3.

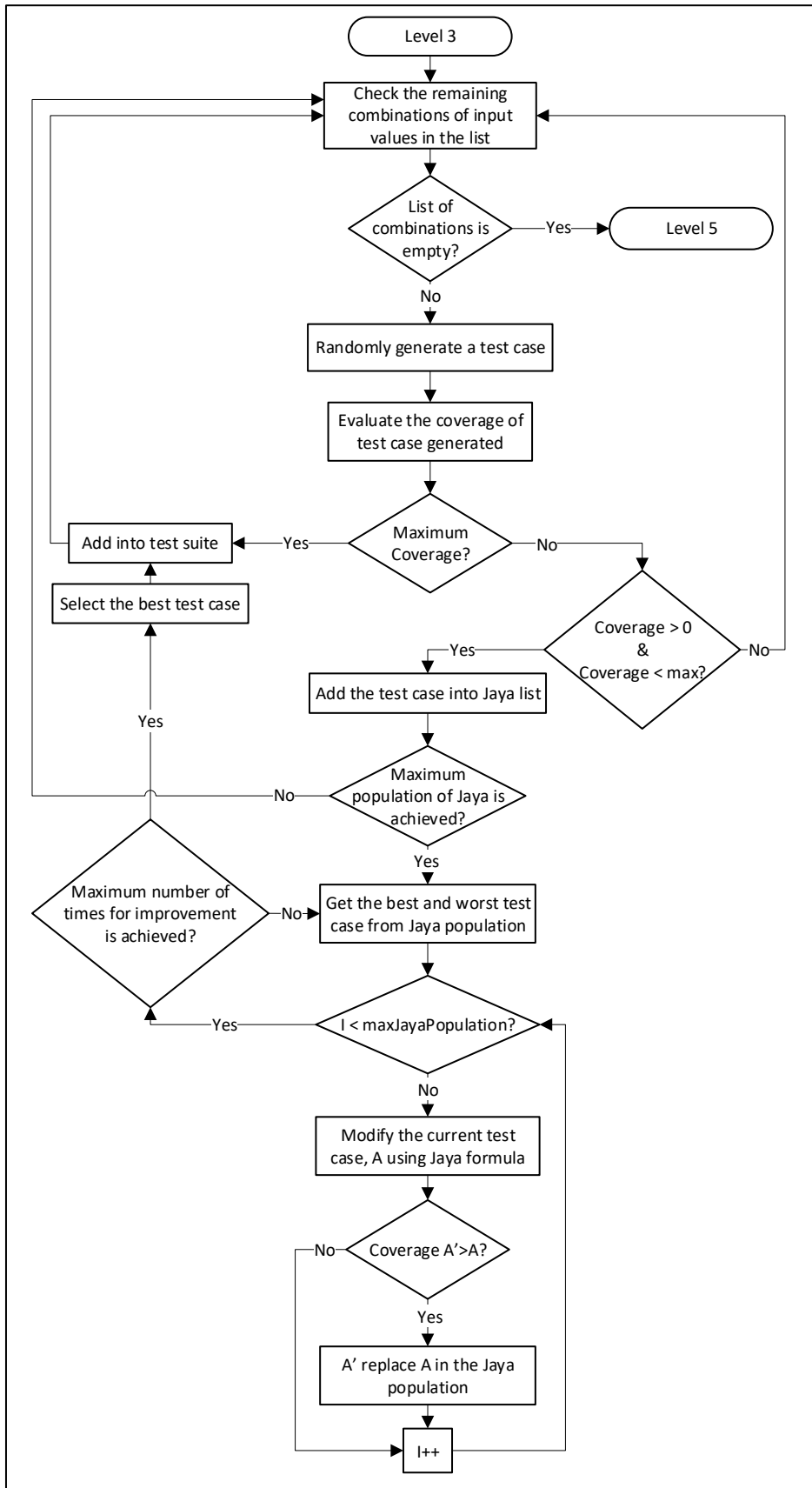


Figure 3.3 The flowchart of test case generation using Jaya algorithm

3.2.2.5 Level 5: Finalization of test suite generation

The test cases generated in step 4 that stored in a temporary test suite is not the final test cases yet. This is because the values that used to represents the real input values are integers. Therefore, the record of data mapping that collected in step 2 is served to revert the integers to its corresponding real input values. Later, the final test suite is generated and it includes the total number of test cases, the input values for each test case, total number of combinations of input values covered and the number of combinations of input values covered by each test case.

3.2.2.6 Graphical user interface design

There are four graphical user interface (GUI) design in this system. The first GUI is for the input of parameters and its values. For the second GUI, it functions to read the input values from file and generate test case through either uniform interaction strength or input-output relationships while the third GUI offers test case generation based on uniform interaction strength which data is from the first GUI. The last GUI serves for the input-output based relation test case generation using the data obtained in first GUI.

The screenshot shows a window titled "Form Preview" with a menu bar containing "Preview" and "Look and Feel". The interface is divided into several sections:

- Parameter Input Section:** Includes a "Parameter Name:" text box, an "Add" button, a "Parameter Added:" list box, and buttons for "Select the parameter in the list to:", "Modify", and "Delete".
- Value Input Section:** Includes a "Value Name:" text box, an "Add" button, a "Value Added:" list box, and buttons for "Select the value in the list to:", "Modify", and "Delete".
- Action Buttons:** A "Load From File" button and a "Finalize Input" button.
- Testing Options:** A section labeled "Types of combinatorial testing:" with two radio buttons: "Uniform Interaction Strength" and "Input-Output Based Relation (IOR)".

Figure 3.4 GUI for the input of parameters and its values

Figure 3.4 shows the first GUI which is the input of parameters and its values. A parameter can be added by entering the parameter name into the text box provided and click Add button. The added parameter is shown in the list and it can modify and delete by clicking the action button. To add the values of each parameter, the parameter in the list has to be selected first and fill in the value's name then click Add button. The added value also can be modified and deleted as well. For all the input of parameters and its values, it will be shown in the text area. After entering all parameters and its values, select Finalize Input to display all data input in table form. Furthermore, two options are available to be chosen to generate the test cases which are through uniform interaction strength or input-output based relation. Once the type of combinatorial testing is selected, new window is shown for further action. If the user wants to enter the data through file reading instead of entering in the provided GUI, select Load From File button to select the file that wish to read and proceed to the next window as shown in Figure 3.5.

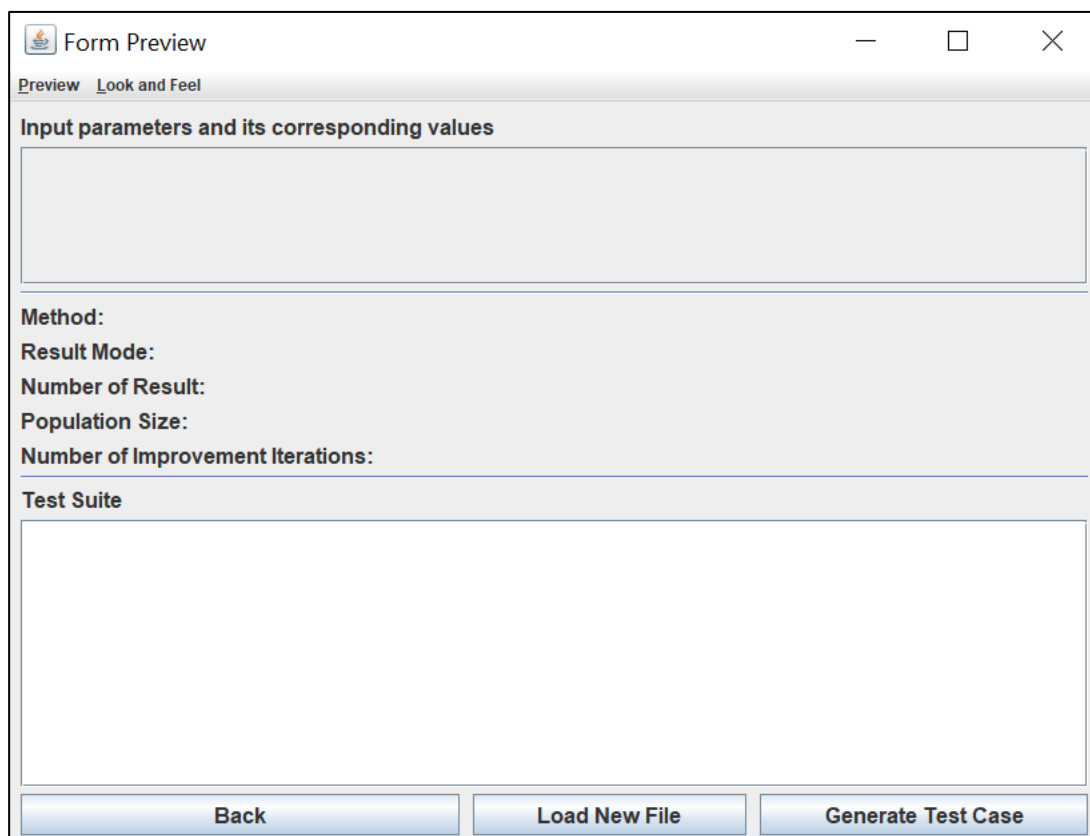


Figure 3.5 GUI of load from file

Once the file is selected to be loaded, the user will be redirected to this window as in Figure 3.5. The input parameters and their corresponding values that stated in the file will be shown in table form on the top of the window. All the information that needed in test case generation are displayed in the respective section to allow user to verify whether the data they input are correct. If they found out there is a mistake in the data entered, they can modify the data in the file and reload the file through Load New File button. If everything is set, they may execute the generation of test case through Generate Test Case button. The Back button can be used by the user to return to previous window.

The screenshot shows a window titled "Form Preview" with a standard Windows-style title bar (minimize, maximize, close buttons). Below the title bar, there are two tabs: "Preview" (selected) and "Look and Feel". The main content area is titled "Uniform Interaction Strength" and contains the following fields:

- Interaction Strength:** A dropdown menu with the value "2" selected.
- Population Size:** An empty text input field.
- Number of Iteration:** An empty text input field.
- Result Mode:** A dropdown menu with the value "Normal" selected.
- Number of Results:** An empty text input field.

Below these fields is a large blue button labeled "Generate Test Case". Underneath the button is a section titled "Output" which contains a large, empty rectangular area for displaying results. At the bottom of the window, there are two buttons: "Back" on the left and "Save Test Suite" on the right.

Figure 3.6 GUI of uniform interaction strength

If uniform interaction strength option is selected in the first GUI, the new window that pop up is shown in Figure 3.6. First, select the interaction strength to determine how the combination is. Only two interaction strength are available which are 2, 3 and 4. Then, enter the population size of test cases that are going to improve, number of improvement iterations, number of result set that wish to have as well as the result mode. There are two result mode to be chosen which are normal mode and best mode. Best mode offers user

to get the best result set out of all result sets generated while normal mode prints all of the result set without suggesting the best one. After that, click Generate Test Case button for test case generation. The test suite will be displayed in the text area provided. To store the test suite generated, click Save Test Suite button. The selection of Back button will back to previous window.

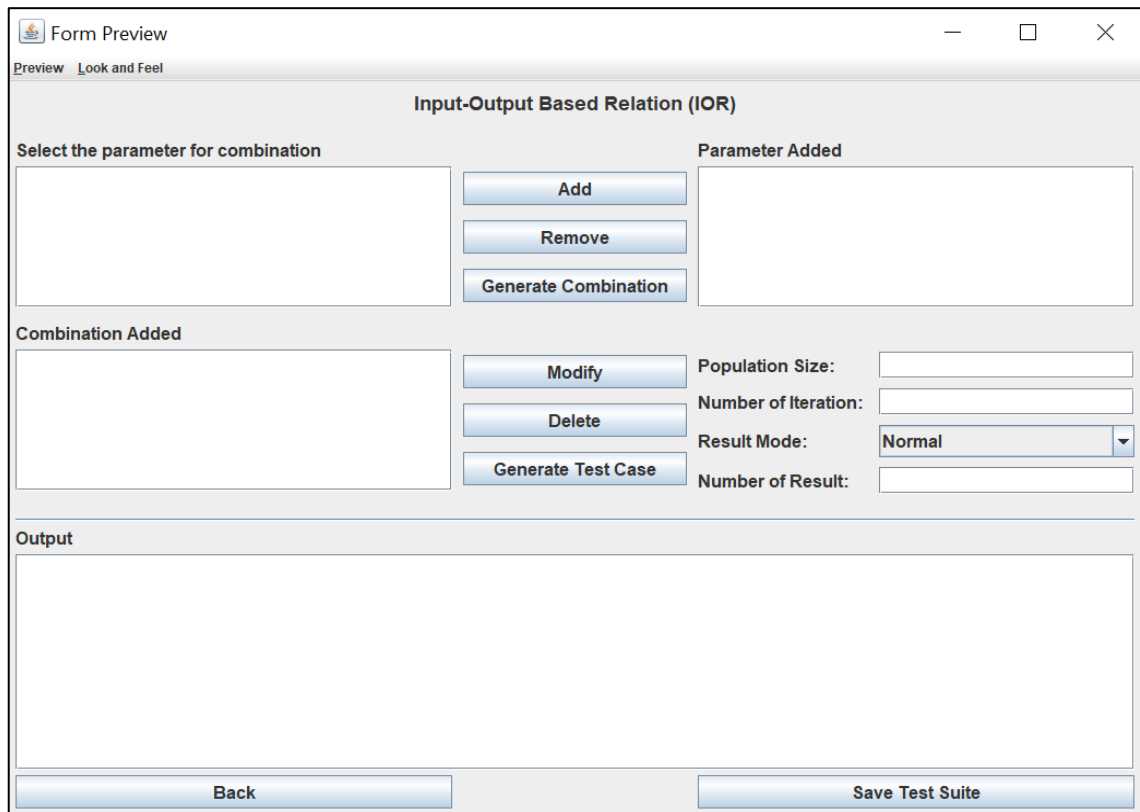


Figure 3.7 GUI of input-output based relation

Figure 3.7 shows the design of GUI of input-output based relation combinatorial testing. It starts with the selection of the desired parameter for combination. The selected parameter is added into the parameter list before the combination of parameters is generated. The parameter in parameter list can be removed through Remove button. The combination of parameter can be generated and saved at the list of combinations through Generate Combination button. The added combinations can be modified and deleted through the selection of respective action button. The additional information that needed to fill in right side of the GUI are same with the third GUI. Test case generation is happened when Generate Test Case button is clicked and the test suite will be displayed in the text area. Save Test Suite button functions to save the result of test suite into a text file while Back button is to go back to the previous window.

3.2.3 Phase 3: Implementation of the Solution

This section explains the design of each level in the system in more details. The pseudocode is used to demonstrate the detail process of each action from Level 1 to 5.

3.2.3.1 Level 1: Reading of input values

There are two parts in reading the data input by the user. The first part is reading the parameters and their corresponding values. It starts with the reading of parameter name and display it on the parameter list. To enter the corresponding value for a parameter, the parameter has to be selected from the parameter list. The entered value will then be displayed on the value list that belongs to the particular parameter. Then, all parameters and their respective values are displayed in a table form for ease of reading. This operation is demonstrated in Figure 3.8.

```
// Read input parameters and their corresponding values
1. BEGIN
2.  READ parameter name
3.  DISPLAY on parameter list
4.
5.  READ corresponding value for the selected parameter
6.  DISPLAY on value list
7.
8.  DISPLAY all parameters and their corresponding values on table form
9. END
```

Figure 3.8 Pseudocode on reading the parameters and their corresponding values

The second part of reading data from user input is to read all necessary information for test case generation which are test case generation method, population size, number of improvement iterations, result mode and number of result set. Firstly, read the test case generation method which is either uniform interaction strength or input-output relationship. If it is uniform interaction strength, read for the interaction strength. Else if it is input-output relationships, read for all the parameter combinations. The process continues with the reading of population size, number of improvement iteration, result mode and number of result sets as shown in Figure 3.9.

```

// Read required information for test case generation
1. BEGIN
2.   READ test case generation method
3.
4.   IF test case generation method is uniform interaction strength
5.     READ interaction strength
6.   ELSEIF test case generation method is input-output relationship
7.     READ input-output relationship combinations
8.   ENDIF
9.
10.  READ population size
11.  READ number of improvement iteration
12.  READ result mode
13.  READ number of result set
14. END

```

Figure 3.9 Pseudocode of reading all necessary information for test case generation

3.2.3.2 Level 2: Data analysis and data mapping

Level 2 begins with the analysis of input data to ensure the data entered is correct in term of data type, no duplication and no empty input. For the input of parameters, no duplication is allowed. All text boxes are not allowed to be empty either to ensure there is no null value is obtained. For any text box that required to enter number, the system will verify the entered value whether is in integer format or not. If the input values are not following the aforementioned criteria, an error message will prompt to alert the users to make the correction. Figure 3.10 is the pseudocode for checking the input data.

```

// Data analysis
1. BEGIN
2.   IF parameter name is duplicated
3.     DISPLAY error message
4.   ENDIF
5.
6.   IF empty string is entered
7.     DISPLAY error message
8.   ENDIF
9.
10.  IF interaction strength or population size or number of improvement
    iteration or number of result set is not integer
11.    DISPLAY error message
12.  ENDIF
13. END

```

Figure 3.10 Pseudocode for data analysis

Next, data mapping process is being carried out. Two arraylist are initialized at the very beginning which serve the purpose of storing the parameters that have paired with a symbolic integer as well as the input values of each parameter which paired with symbolic integer and symbolic integer of its corresponding parameter. The mapping process is initiated by mapping of symbolic integer with parameter. Looping takes place to go through every parameter and assign the symbolic integer to each parameter. The input values of each parameter undergo the same process but the only difference is the symbolic integer of the parameter also includes inside the input values' arraylist. The pseudocode of data mapping is shown in Figure 3.11.

```

// Data mapping
1. BEGIN
2.   INITIALIZE an arraylist to store mapped parameters
3.   INITIALIZE an arraylist to store mapped values
4.
5.   FOR i = 0 to total number of parameters
6.     SET the key for parameter
7.     ADD the key and the name of parameter in paraList
8.
9.     FOR j = 0 to total number of values for the selected parameter
10.      SET the key for input value
11.      ADD the key, the name of input value and parameter key in valueList
12.    ENDFOR
13.  ENDFOR
14. END

```

Figure 3.11 Pseudocode for data mapping

3.2.3.3 Level 3: Combinations of input values generation

There are two types of combination of input values which are combination based on uniform interaction strength and input-output based relation. The pseudocode of each type of combination is shown in Figure 3.12 and 3.13 respectively.

For combination based on uniform strength interaction, the first step is to get the interaction strength. Next, two arraylist which used to store the combinations of parameters as well as the input values that are going to be generated are defined. If the interaction strength is equal to two, the iteration for generating combination between parameters and combinations between input values will go through twice respectively. The combinations are obtained through the non-overlapping concatenation between parameters and input values. The result of combinations is then saved in the arraylist that

defined initially. If the entered interaction strength is three, the process is gone through as stated above with the only difference is the number of loops to generate the combinations is three times.

Moreover, generation of combination using input-output based relation is commenced by initializing two arraylist. The first arraylist is used to store the combination of parameter which is in the symbolic form while second arraylist is a temporary list for the input values based on selected parameter's combination. Later, the number of combinations of parameters is acquired and each parameter in the combination is converted to symbolic form and added into the first arraylist. Based on the parameters' combinations in first arraylist, the input values that correspond to the parameter is saved into the second arraylist. Each of the input values goes through combination by concatenating with each other without repetition based on the number of parameters in each combination of parameters. The second arraylist will be cleared once the combination of input values of selected parameters' combination is done in order to allow the next combination of input values to be happened.

```

// Combination of parameters and values for uniform strength interaction
1. BEGIN
2.   GET interaction strength
3.
4.   INITIALIZE paraCombinationList arraylist to store the combination of
   parameter
5.   INITIALIZE valueCombinationList arraylist to store the combination of
   value
6.
7.   IF interaction strength = 2
8.     // To generate the combinations between parameters
9.     FOR i = 0 to size of paraList
10.      FOR j = i + 1 to size of paraList
11.        CONCATENATE current paraList array with next paraList array
12.        ADD into paraCombinationList
13.      ENDFOR
14.    ENDFOR
15.
16.    // To generate the combinations between values
17.    FOR i = 0 to size of valueList
18.      FOR j = i + 1 to size of valueList
19.        CONCATENATE current valueList array with next valueList array
20.        ADD into valueCombinationList
21.      ENDFOR
22.    ENDFOR
23.
24.  ELSE IF interaction strength = 3
25.    // To generate the combinations between parameters
26.    FOR i = 0 to size of paraList
27.      FOR j = i + 1 to size of paraList
28.        FOR k = j + 1 to size of paraList
29.          CONCATENATE current paraList array with next paraList array and
          subsequent paraList array
30.          ADD into paraCombinationList
31.        ENDFOR
32.      ENDFOR
33.    ENDFOR
34.
35.    // To generate the combinations between values
36.    FOR i = 0 to size of valueList
37.      FOR j = i + 1 to size of valueList
38.        FOR k = j + 1 to size of valueList
39.          CONCATENATE current valueList array with next valueList array
          and subsequent valueList array
40.          ADD into paraCombinationList
41.        ENDFOR
42.      ENDFOR
43.    ENDFOR
44.  ENDIF
45. END

```

Figure 3.12 Pseudocode of combination of input values generation for uniform strength interaction

```

// Combination of parameters and input values for input-output relationship
1. BEGIN
2.   INITIALIZE paraKeyCombinationList arraylist
3.   INITIALIZE tempValueList arraylist
4.
5.   READ total number of combinations
6.
7.   // Get the combinations
8.   FOR i = 0 to total number of combinations
9.     READ the combination
10.    CONVERT the combination into the form of parameter key
11.    ADD into paraKeyCombinationList
12.  ENDFOR
13.
14.  // Generate the combinations of input value of IOR
15.  FOR i = 0 to size of paraKeyCombinationList
16.
17.    // Get the input values for each parameter in the IOR combinations
18.    FOREACH parameter in paraKeyCombinationList
19.      GET the input value from corresponding parameter
20.      ADD in tempValueList
21.    ENDFOREACH
22.
23.    // Get the combinations of input values for that length equal to 2
24.    IF the length of selected paraKeyCombinationList is equal to 2
25.      // To generate the combinations between values
26.      FOR i = 0 to size of tempValueList
27.        FOR j = i + 1 to size of tempValueList
28.          CONCATENATE current tempValueList with next tempValueList array
29.          ADD in valueCombinationList
30.        ENDFOR
31.      ENDFOR
32.
33.      // Get the combinations of input values that length equal to 3
34.      ELSE IF the length of selected paraKeyCombinationList is equal to 3
35.        FOR i = 0 to size of tempValueList
36.          FOR j = i + 1 to size of tempValueList
37.            FOR k = j + 1 to size of tempValueList
38.              CONCATENATE current tempValueList array with current
39.                tempValueList array and subsequent tempValueList array
40.              ADD into valueCombinationList
41.            ENDFOR
42.          ENDFOR
43.        ENDFOR
44.      ENDFOR
45.      // Clear the tempValueList arraylist for the next combination
46.      CLEAR tempValueList
47.    ENDFOR
48.  END

```

Figure 3.13 Pseudocode of combination of input values generation for input-output relationship

3.2.3.4 Level 4: Test case generation based on Jaya algorithm

After the combinations of input values are generated in Level 3, the process is continued with test case generation using Jaya algorithm. Firstly, get the boundary values for the values in each parameter and the arraylist that stores all value combinations. Then, initialize two arraylist to store the test suite and the test cases that are needed to be improved. The looping starts by checking whether the population size of test cases that need to be improved is reaching the maximum population size defined by user. If the size is not yet reached what user specified, a random test case is generated else proceed with improvement of the test cases. The generated random test case is then examined and identify the number of value combinations covered. If it has maximum coverage, the test case will be added into test suite and the value combinations covered by this test case are removed from the arraylist that stores all value combinations. However, if the random test case's coverage neither maximum nor zero, this test case will be added into the list that stores all test cases that need to be improved.

Test case improvement using Jaya algorithm begins when the maximum population size is achieved. The number of improvements is get based on what user defined and the following process is looping based in the number of improvements. Initially, get the best and the worst test case from the population. The population mentioned is the list of all test cases that are needed to be improved. Next, initialize two random number for the calculation later. Every test case in the population will go through the improvement using Jaya algorithm. For each value that represents each parameter, it will be modified using Jaya formula as stated in line 46 in the pseudocode. After the calculation made, the modified value must be checked using boundary value to prevent the value key from exceeding the permitted range. If the modified value is lower than lower boundary value, it will be replaced with the lower boundary value while the modified value that excess higher boundary value is replaced by the higher boundary value. Later, modified test case will replace unmodified test case if its number of combinations covered is higher than latter. After the improvements iterate based on the number of times user defined, the best test case which has the highest coverage will be selected, added into the test suite and the combinations that covered by this test case are removed. This whole process is kept on looping until all value combinations are covered. The pseudocode for this process is shown in Figure 3.14 and Figure 3.15.

```

// Test case generation using Jaya algorithm
1. BEGIN
2. GET the boundary value for each random test case that store in
3. GET valueCombinationList arraylist
4.
5. INITIALIZE testSuite arraylist
6. INITIALIZE toBeImprovedTestCases arraylist
7.
8. //The process is continued until all values combination are fully covered
9. DO
10. IF size of toBeImprovedTestCases is equal to maximum population size
11.
12. GENERATE a random test case using the values in each parameter
13.
14. // Check the number of combinations covered
15. FOREACH valueCombination in valueCombinationList
16. IF the values in random test case is equal to valueCombination
17. INCREASE the number of combinations covered by 1
18. ENDFOR EACH
19. ENDFOREACH
20.
21. // Add the test case into test suite if the coverage is maximum
22. IF the number of combinations covered by current test case is equal
to maximum number of test case covered
23. ADD current random test case into test suite
24. REMOVE the combination covered by current random test case
25.
26. // Add those test cases that has coverage lower than maximum but
it is not zero into test case that needed to improve
27. ELSE IF the number of combinations covered by current test case is
larger than 0 but smaller than maximum coverage
28. ADD random test case and its coverage into toBeImprovedTestCases
29.
30. ELSE
31. // Apply Jaya algorithm when the maximum population size is achieved
32.
33. GET boundary value of each parameter
34.
35. FOR i = 0 to number of improvement iteration
36.
37. GET the best coverage test case from the population
38. GET the worst coverage test case from the population
39.

```

Figure 3.14 Pseudocode for test case generation using Jaya algorithm (Part 1)


```

40.     INITIALIZE two random numbers (r1 and r2)
41.
42.     FOREACH toBeImprovedTestCase in toBeImprovedTestCases
43.
44.         // Improve the test case by modifying with Jaya formula
45.         FOREACH value in toBeImprovedTestCase
46.             modifiedValue = value in toBeImprovedTestCase + r1 * (value
                in best test case - value in toBeImprovedTestCase)
                - r2 * (value in best test case - value in
                toBeImprovedTestCase)
47.
48.             IF modifiedValue is lower than lowerBoundaryValue
49.                 REPLACE modifiedValue with lowerBoundaryValue
50.             ELSEIF modifiedValue is higher than higherBoundaryValue
51.                 REPLACE modifiedValue with higherBoundaryValue
52.             ENDIF
53.         ENDFOREACH
54.
55.         IF coverage of modifiedTestCase is higher than coverage of
            toBeImprovedTestCase
56.             REPLACE toBeImprovedTestCase with modifiedTestCase
57.         ENDIF
58.     ENDFOREACH
59.
60.     GET the best coverage test case from the population
61.     ADD the best coverage test case into testSuite
62.     REMOVE the valueCombination covered by the best coverage test
    case
63.     ENDIF
64. ENDIF
65. WHILE (valueCombinationList is not empty)
66. END

```

Figure 3.15 Pseudocode for test case generation using Jaya algorithm (Part 2)

3.2.3.5 Level 5: Finalization of test case generation

The test suite generated in Level 4 is not the final test suite as the value for each parameter in the test case is still in symbolic form. Therefore, it must convert back to the name that user input earlier. Firstly, the input values of each of the test case inside the test suite will retrieve their corresponding name based on their symbolic value and replace the symbolic value with the name obtained. Later, the test case with the real input values will be added into the final test suite. After this, the final test suite and other information are displayed to the user. Figure 3.16 is the pseudocode for the finalization of test case generation.

```

// Finalization of test case generation
1. BEGIN
2.
3. INITIALIZE finalTestSuiten arraylist
4.
5. // Convert the value which is integer form to the original form
6. FOR i = 0 to size of testSuite
7.   FOR j = 0 to size of test case
8.     GET the name for current value based on its key
9.     REPLACE the key with its corresponding name
10.  ENDFOR
11.
12.  ADD into finalTestSuite
13. ENDFOR
14.
15. PRINT the finalTestSuite
16. END

```

Figure 3.16 Pseudocode of finalization of test case generation

3.2.4 Phase 4: Test and Evaluation

In this phase, experiments are conducted to make a comparison between existing input-output based relation and uniform strength interaction combinatorial testing that applying different types of approaches with the proposed solution which used Jaya algorithm in handling the combinatorial optimization.

3.2.4.1 Experiments for input-output based relation combinatorial testing

The existing input-output based relation approaches that are involving in these experiments are Density (Z. Y. Wang et al., 2008), TVG (Arshem, 2009), ReqOrder (Ziyuan et al., 2007), ParaOrder (Z. Y. Wang et al., 2008), Union (Patrick J. Schroeder & Korel, 2000), Greedy (P. J. Schroeder et al., 2002), ITTDG (Othman & Zamli, 2011) and AURA (Ong & Zamli, 2011). The attribute that utilized in determining the efficiency of the strategies is the number of test cases generated based on the same scenario. There are two experiments will be carried out to complete the testing. The aforementioned combinatorial strategies, system configuration and result of both experiments are adopted from the study which are in published (AbdulRahman A Alsewari et al., 2015). The first experiment will be using 10 parameters and each parameter has 3 input values while the second experiment consists of 3 parameters with 2 input values each, 3 parameters with 3 input values each, 3 parameters with 4 input values each and 1 parameter with 5 input values each. The parameters are labelled from 0 to 9 for both experiments. There are 60 input-output relationships (R) that defined for both experiments as shown in Table 3.2.

Table 3.2 60 input-output relationships (R) that utilized in experiments

	10th relationship	20th relationship	30th relationship
Relationship (R)	{1, 2, 7, 8}	{2, 3, 4, 8}	{1, 3, 6, 9}
	{0, 1, 2, 9}	{2, 3, 5}	{2, 4, 7, 8}
	{4, 5, 7, 8}	{5, 6}	{0, 2, 6, 9}
	{0, 1, 3, 9}	{0, 6, 8}	{0, 1, 7, 8}
	{0, 3, 8}	{8, 9}	{0, 3, 7, 9}
	{6, 7, 8}	{0, 5}	{3, 4, 7, 8}
	{4, 9}	{1, 3, 5, 9}	{1, 5, 7, 9}
	{1, 3, 4}	{1, 6, 7, 9}	{1, 3, 6, 8}
	{0, 2, 6, 7}	{0, 4}	{1, 2, 5}
	{4, 6}	{0, 2, 3}	{3, 4, 5, 7}
	40th relationship	50th relationship	60th relationship
	{0, 2, 7, 9}	{2, 3, 9}	{0, 6, 7, 9}
	{1, 2, 3}	{1, 5, 8}	{2, 6, 7, 9}
	{1, 2, 6}	{1, 3, 5, 7}	{2, 6, 8}
	{2, 5, 9}	{0, 1, 2, 7}	{2, 3, 6}
	{3, 6, 7}	{2, 4, 5, 7}	{1, 3, 7, 9}
	{1, 2, 4, 7}	{1, 4, 5}	{2, 3, 7}
	{2, 5, 8}	{0, 1, 7, 9}	{0, 2, 7, 8}
	{0, 1, 6, 7}	{0, 1, 3, 6}	{0, 1, 6, 9}
	{3, 5, 8}	{1, 4, 8}	{1, 3, 7, 8}
	{0, 1, 2, 8}	{3, 5, 7, 9}	{0, 1, 3, 7}

The configuration of first experiment is IOR (N, 3^{10} , R) where N is the number of test case for this configuration while R is the input-output relationships as listed in Table 3.2. 10 parameters that designated from 0 to 9 and each of them consists 3 input values are going to undergo combinatorial testing for six iterations. The R will be the first 10 relationships from Table 3.2 for iteration 1. For the subsequent iterations, the next 10 relationships will be added into the experiment until all 60 relationships are being tested. The result of first experiment of existing IOR strategies is listed in Table 3.3.

Table 3.3 The size of test suite of existing IOR strategies using configuration IOR (N, 3^{10} , R)

R	Density	TVG	ReqOrder	ParaOrder	Union	Greedy	ITTDG	AURA
10	86	86	153	105	503	104	81	89
20	95	105	148	103	858	110	94	99
30	116	125	151	117	1599	122	114	132
40	126	135	160	120	2057	134	122	139
50	135	139	169	148	2635	138	131	147
60	144	150	176	142	3257	143	141	158

For the second experiment, the configuration is set to be IOR ($N, 2^3 3^3 4^3 5^1, R$). R is the input-output relationships in Table 3.2 while N is the number of test cases. In second experiment, there will be 3 parameters with 2 input values for each parameter, 3 parameters with 3 input values for each parameter, 3 parameters with 4 input values for each parameter and 1 parameter with 5 input values for each parameter. The total number of parameters in this experiment is 10 and each parameter is tagged from 0 to 9. The experiment will be conducted for six iterations. R will increment in each iteration where first 10 relationships is evaluated in first iteration and the following relationships are added in the successive iterations until all relationships are assessed. Table 3.4 shows the result of current IOR approaches using second experiment configuration.

Table 3.4 The size of test suite of existing IOR strategies using configuration ($N, 2^3 3^3 4^3 5^1, R$)

R	Density	TVG	ReqOrder	ParaOrder	Union	Greedy	ITTDG	AURA
10	144	144	154	144	505	137	144	144
20	160	161	187	161	929	158	160	182
30	165	179	207	179	1861	181	169	200
40	165	181	203	183	2244	183	173	207
50	182	194	251	200	2820	198	183	222
60	197	209	250	204	3587	207	199	230

3.2.4.2 Experiment for uniform interaction strength combinatorial testing

Uniform interaction strength combinatorial testing is widely implemented using different strategies which including Harmony Search Strategy (HSS), Simulated Annealing (SA), Genetic Algorithm (GA), Ant Colony Algorithm (ACA), Automatic Efficient Test Generator (AETG), IPOG, Jenny, TVG and Particle Swarm Test Generator (PSTG). The result and system configuration of all above strategies are being published in (Abdulrahmn A. Alsewari & Zamli, 2011). Hence, the configuration of the experiment is derived from the research above. One experiment with fourteen configurations will be conducted to evaluate the size of test suite generated.

Table 3.5 System configuration for uniform interaction strength experiment

No.	Interaction Strength (t)	System Configuration
C 1	2	CA (N; 2, 4, 3)
C 2	2	CA (N; 2, 13, 3)
C 3	2	CA (N; 2, 10, 10)
C 4	2	CA (N; 2, 10, 15)
C 5	2	CA (N; 2, 10, 5)
C 6	3	CA (N; 3, 6, 3)
C 7	3	CA (N; 3, 6, 4)
C 8	3	CA (N; 3, 6, 5)
C 9	3	CA (N; 3, 6, 6)
C 10	3	CA (N; 3, 7, 5)
C 11	2	MCA (N; 2, 11, (5, 3 ⁸ , 2 ²))
C 12	2	MCA (N; 2, 20, (7, 6, 5, 4 ⁶ , 3 ⁸ , 2 ³))
C 13	3	MCA (N; 3, 6, (5 ² , 4 ² , 3 ²))
C 14	3	MCA (N; 3, 7, (10, 6 ² , 4 ³ , 3))

Table 3.5 is the system configuration for each test in the experiment. CA is the covering array while MCA is the mixed covering array as discussed in Chapter 2.1. There are 14 tests that are going to conduct, evaluate and compare the performance with other aforementioned strategies that applied uniform interaction strength. The result of those strategies is presented in Table 3.6.

Table 3.6 The test suite size of existing strategies using uniform interaction strength

System Configuration	HSS	SA	GA	ACA	AETG	IPOG	Jenny	TVG	PSTG
C 1	9	9	9	9	9	9	10	11	9
C 2	18	16	17	17	17	20	20	19	17
C 3	155	NA	157	159	NA	176	157	208	NA
C 4	341	NA	NA	NA	NA	373	336	473	NA
C 5	43	NA	NA	NA	NA	50	45	51	45
C 6	39	33	33	33	38	53	51	49	42
C 7	70	64	64	64	77	64	112	123	102
C 8	199	152	125	125	194	216	215	234	NA
C 9	336	300	331	330	330	382	373	407	338
C 10	236	201	218	218	218	274	236	271	229
C 11	20	15	15	16	20	19	23	22	NA
C 12	48	42	42	42	44	43	50	51	48
C 13	119	100	108	106	114	111	131	136	NA
C 14	378	360	360	361	377	383	399	414	385

Note: NA represents Not Available

After conducting the experiments using Jaya algorithm optimization strategy, comparison between proposed strategy and the existing IOR as well as uniform interaction strength strategies can be takes place to identify the effectiveness of suggested approach in IOR and uniform interaction strength combinatorial testing. The strategy with the least number of test cases in experiments is considered as the most efficient approach in handling IOR and uniform interaction strength combinatorial testing. The discussion of the result obtained is further conferred.

3.2.5 Phase 5: Documentation

Documentation is a continuous process throughout the project to come out with an entire thesis. During phase 1, all the findings which get from literature review is being analysed and recorded. The overview and detail design of input-output based relation combinatorial testing strategy using Jaya algorithm in phase 2 and 3 respectively are filed in proper way. Moreover, the settings of the experiments and the result of existing strategies of both experiments are listed for testing purpose. After gathering all information from each phase, the documentation is then finalized and applied the right formatting.

3.3 HARDWARE AND SOFTWARE

This section explains the usage of hardware and software in this project. Table 3.7 lists the hardware that will be used while Table 3.8 shows the needed software to develop the suggested IOR combinatorial testing strategy and document the findings.

Table 3.7 List of needed hardware

Hardware	Purpose
Laptop	Workstation of this project
Printer	Device for printing required documents

Table 3.8 List of needed software

Software	Purpose
Windows 10 Professional	Operating system of the workstation
Microsoft Word 2016	To do all documentations in this project
Microsoft Visio Professional 2016	To draw required figures and flowcharts
Microsoft Project Professional 2016	To construct the Gantt chart
IntelliJ IDEA Community Edition 2018.3	To develop the proposed CTJ system
Notepad ++	To create a text file for the parameters and its input values for IOR combinatorial testing

3.4 GANTT CHART

A Gantt chart has been illustrated to show the progress of the project based on the planning of research methodology. The Gantt chart is attached in Appendix A for reference.

CHAPTER 4

IMPLEMENTATION, RESULT AND DISCUSSION

4.1 INTRODUCTION

This chapter discusses the implementation of CTJ in real environment, the result of implementation of CTJ and discussion about the result obtained. The implementation of input-output based combinatorial testing strategy using Jaya algorithm will be explained thoroughly in the first section. Moreover, section two shows the results of execution of CTJ and the discussion of CTJ's performance compared to other existing strategies.

4.2 IMPLEMENTATION OF CTJ

There are five main steps in generating test suite using CTJ which are reading of input values, data analysis and data mapping, combinations of input values generation, test case generation based on Jaya algorithm as well as finalization of test suite generation. The details of implementation of each step will be explained in the following subsections.

4.2.1 Level 1: Reading of input values

There are two ways to get the user input in CTJ. The first way is through entering the data in the provided GUI while the second way is through file reading of the data. Both ways to get the data required are demonstrated as below.

Initially, the user is needed to enter the input parameters and corresponding values in the textboxes provided for the first way of reading the inputs. The input parameters and its values can be modified and deleted when needed. Any modification and deletion

of input parameters and corresponding values can be made by selecting the modify and delete button respectively. Figure 4.1 shows the example of completed input parameters and its corresponding values.

The screenshot shows a web application interface titled "Home". It is divided into several sections for configuring input parameters and values.

Parameter Configuration Section:

- Parameter Name:** An empty text input field.
- Parameter Added:** A list box containing "CPU", "GPU", and "RAM". "CPU" is currently selected.
- Select the parameter in the list to:** A label above three buttons: "Add", "Modify", and "Delete".

Value Configuration Section:

- Please select the parameter from the list to add its corresponding values:** A label above the section.
- Value Name:** An empty text input field.
- Value Added:** A list box containing "Intel", "AMD", and "Qualcomm".
- Select the value in the list to:** A label above three buttons: "Add", "Modify", and "Delete".

File Loading Section:

- Load From File:** A large blue button.

List of parameter and its values entered:

CPU	GPU	RAM
Intel	Nvidia	Kingston
AMD	AMD	Samsung
Qualcomm	Asus	Micron

Finalize Input: A large blue button.

Types of combinatorial testing:

- Uniform Interaction Strength** (selected)
- Input-Output Based Relation (IOR)**

Figure 4.1 Example of completed input parameters and its corresponding values at the Home page of CTJ

For reading the input through file reading, it has to be triggered by selecting the “Load From File” button in the Home page. Then, select the configuration file that would like to be utilized in the test case generation as shown in Figure 4.2. The data in the file is displayed at the respective section as in Figure 4.3. If the information entered is founded to be an error, user can replace current file with a new file by selecting “Load New File” button.

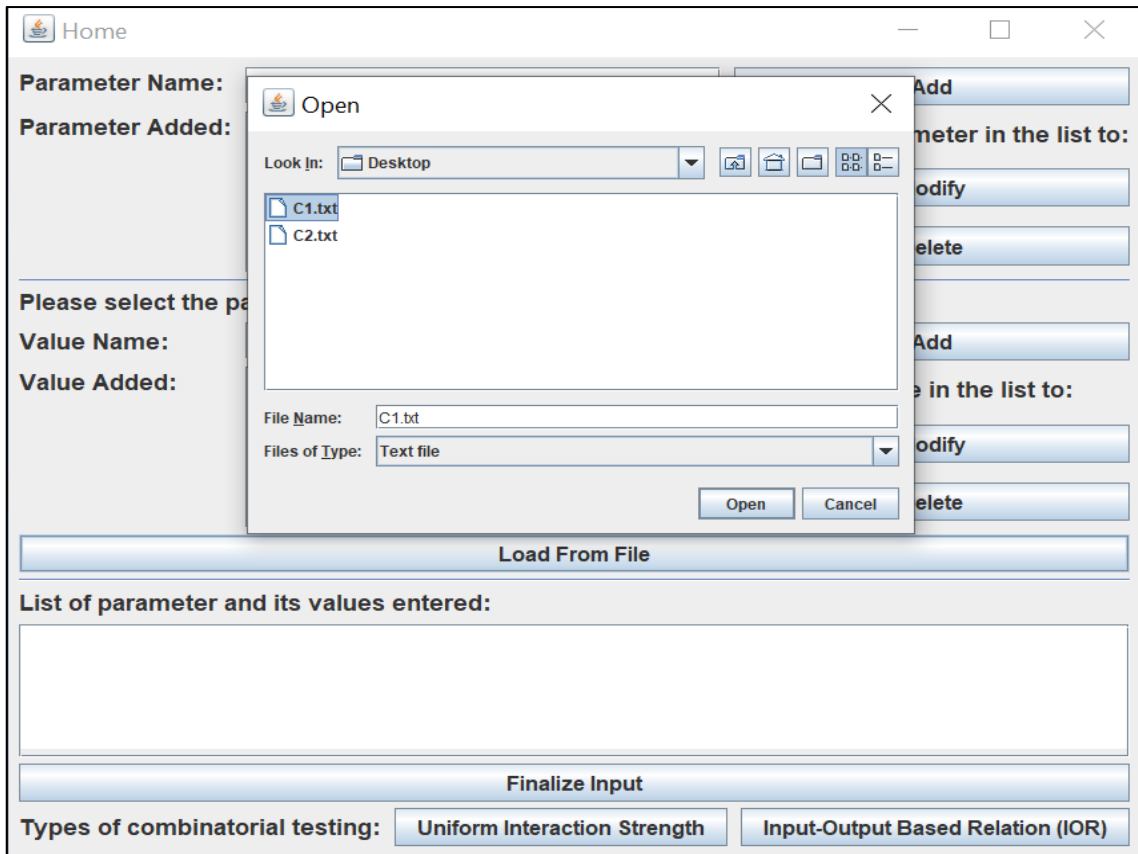


Figure 4.2 File selection for Load From File option

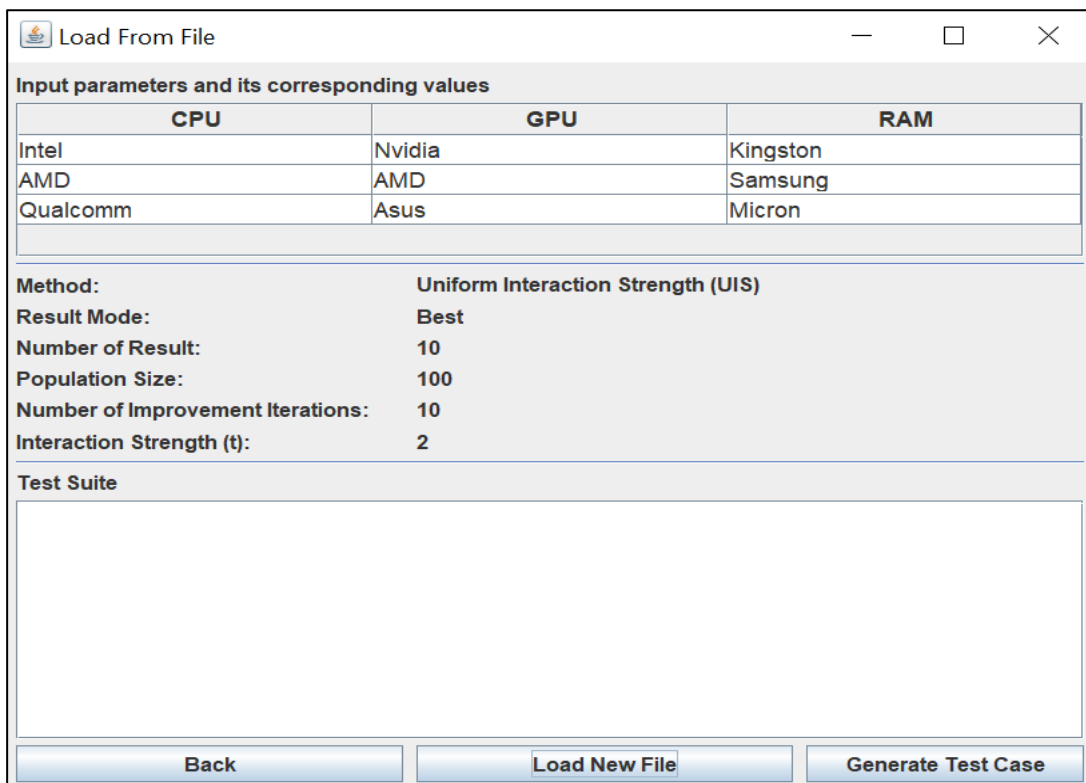


Figure 4.3 The GUI after reading the data from file

4.2.2 Level 2: Data analysis and data mapping

Data analysis process is happened when user enters the parameter and its corresponding values. Any duplications of parameters or input-output relationships are prohibited. Thus, an error message will pop up if it happened as shown in Figure 4.4.

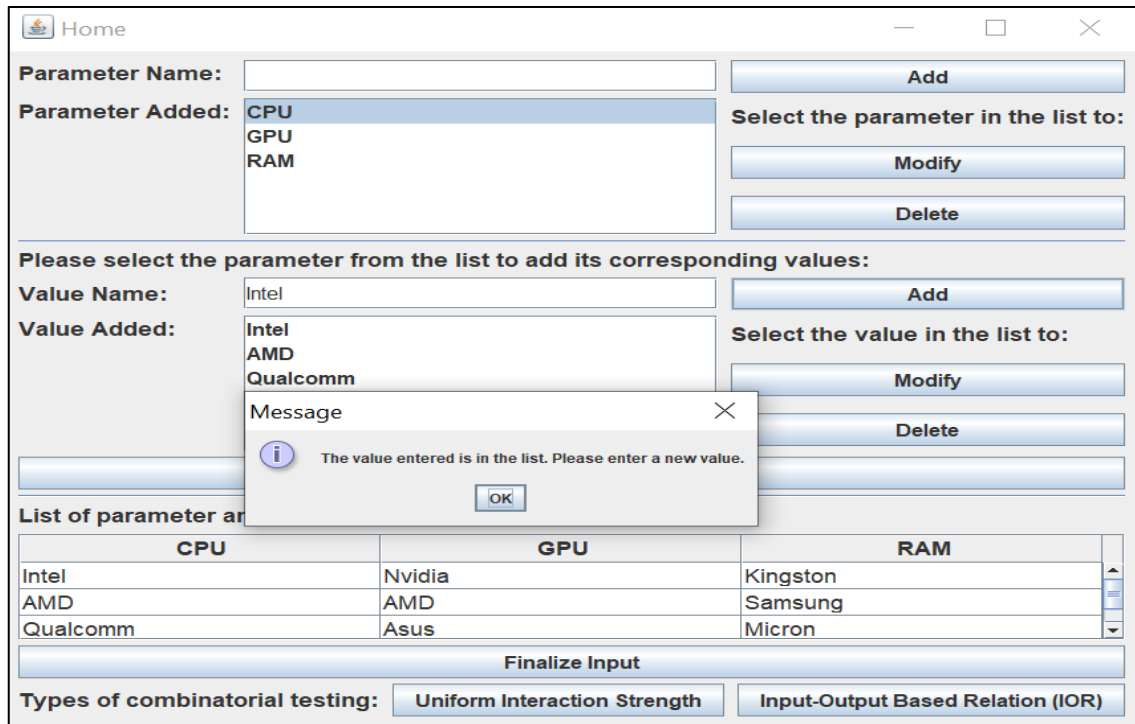


Figure 4.4 Error message for duplication of value entered

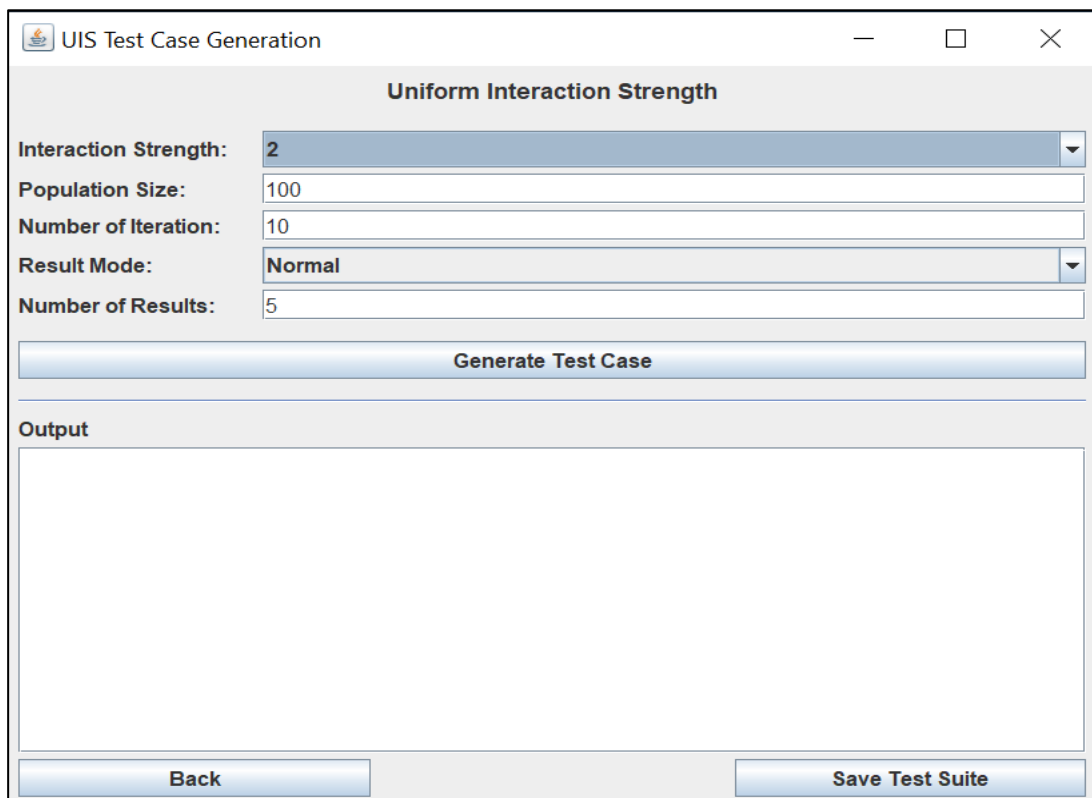
Further, all input parameters and its corresponding values will undergo data mapping to increase the performance of CTJ. Taking the example from the demonstration above, the outcome of data mapping is shown in Table 4.1.

Table 4.1 The mapped values for both input parameters and their corresponding values

Input parameters	Mapped values for input parameters	Corresponding values	Mapped values for corresponding values
CPU	0	Intel	0
		AMD	1
		Qualcomm	2
GPU	1	Nvidia	3
		AMD	4
		Asus	5
RAM	2	Kingston	6
		Samsung	7
		Micron	8

4.2.3 Level 3: Combinations of input values generation

Next, user is required to select whether to generate the test cases through uniform interaction strength or input-output based relation. If uniform interaction strength option is chosen, user will be redirected to select the interaction strength, population size, number of improvement iterations, result mode and number of result set that wish to get. The interaction strengths supported are 2 to 4 and there are two result mode which are normal and best. Best mode represents user defines the number of result set are generated and the best out all generated result set will be displayed at the end of the text area while normal mode is displaying the result without displaying the best result set. The example of this operation is shown in Figure 4.5.



The screenshot shows a window titled "UIS Test Case Generation" with a subtitle "Uniform Interaction Strength". The window contains the following fields and controls:

- Interaction Strength:** A dropdown menu with the value "2" selected.
- Population Size:** A text input field containing "100".
- Number of Iteration:** A text input field containing "10".
- Result Mode:** A dropdown menu with the value "Normal" selected.
- Number of Results:** A text input field containing "5".

Below these fields is a large blue button labeled "Generate Test Case". Underneath the button is a large empty text area labeled "Output". At the bottom of the window are two buttons: "Back" on the left and "Save Test Suite" on the right.

Figure 4.5 The complete details that have to be filled in before test case generation through uniform interaction strength

If the user wants to go for input-output based relation, this option must be selected to redirect to another user interface for entering the details of relationships. The relationships are generated by selecting the desired parameters from the list of parameters and added it into the list of all IOR relationships. This process is demonstrated in Figure 4.6.

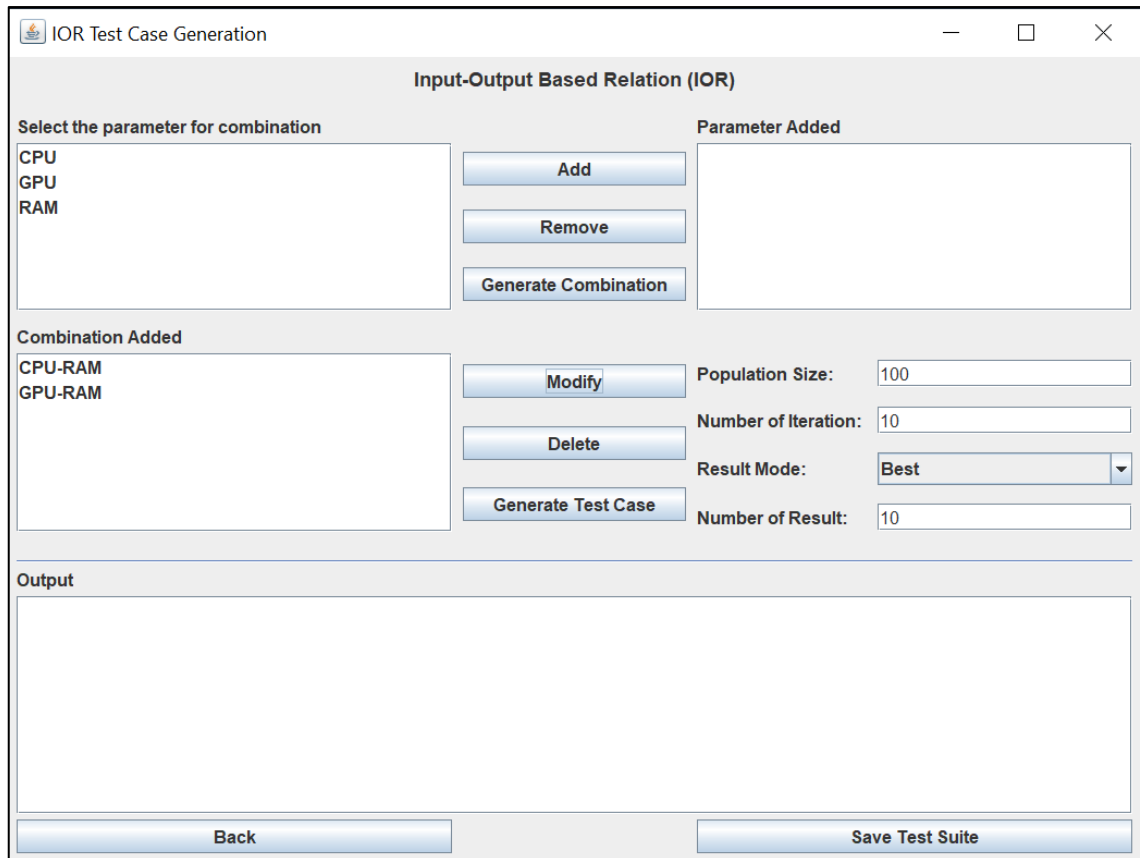


Figure 4.6 The complete details that have to be filled in before test case generation through input-output relationships

Then, the test suite will be generated once “Generate Test Cases” button is selected.

4.2.4 Level 4: Test case generation based on Jaya algorithm

Once “Generate Test Cases” button is clicked, the test case generation process will begin in background. The details of the execution can refer to section 3.2.3.4. The test suite will be generated when the test case generation is completely done. Figure 4.7 and Figure 4.8 show the example of test suite generated based on the configuration above.

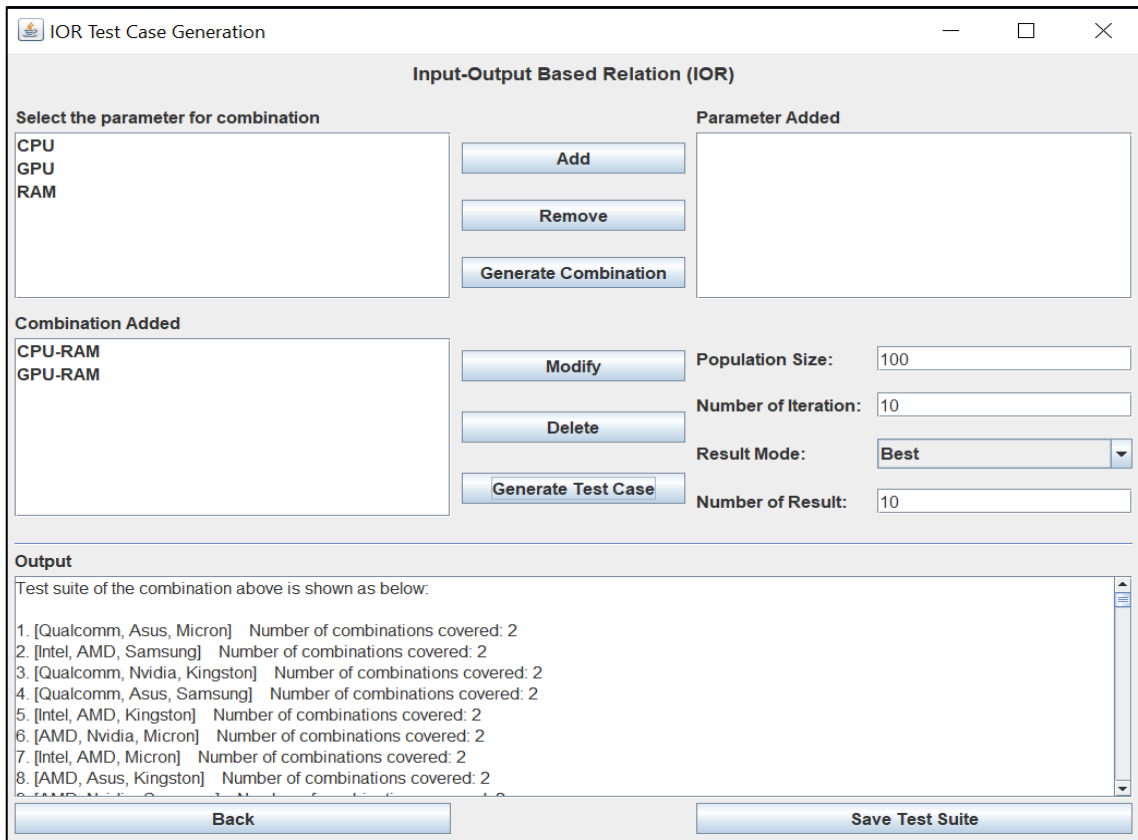


Figure 4.7 The test suite generated through ordinary GUI input

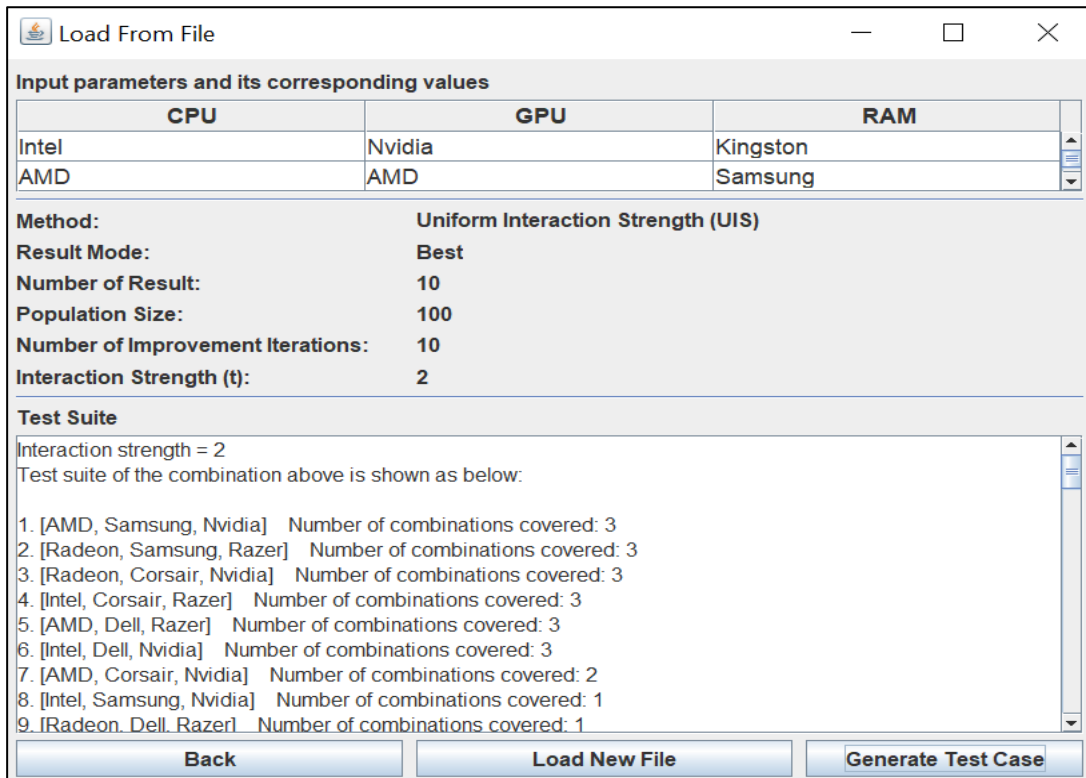


Figure 4.8 Test suite generated through Load From File

4.2.5 Level 5: Finalization of test suite generation

Reverse data mapping happens once the test cases are completely generated. It will revert the mapped values that used in data combination and test case generation into the original string. After that, the complete test suite will be printed on the text area provided as shown in Figure 4.7 and Figure 4.8.

4.3 EXPERIMENTAL RESULTS AND DISCUSSION

This section is divided into three parts which are parameter tuning of CTJ, two experiments for IOR and one experiment for uniform strength interaction strength to evaluate the difference between CTJ with other existing strategy in handling combinatorial optimization.

4.3.1 Parameter tuning of CTJ

All experiments conducted are using Intel i7-6500U as the CPU with the RAM of 8GB in Windows 10 Professional operating system. There are only two common controlling parameters involved in CTJ which are population size and number of iterations. Hence, tuning of parameters setting is performed to ensure the optimal results and efficiency of CTJ before the experiments are carried out. The tuning process is executed using one of the configurations from each IOR and uniform interaction strength experiments in (AbdulRahman A Alsewari et al., 2015) and (Abdulrahmn A. Alsewari & Zamli, 2011). Three types of parameters settings that stated in Table 4.2 are being experimented for 10 iterations to figure out which parameters setting will generate the most optimum and efficient result.

Table 4.2 Parameter Setting

Parameters Setting	Population Size	Number of Iterations
S1	10	100
S2	50	500
S3	100	1000

Table 4.3 Test case size and execution time in 30 input-output relationships configuration

Parameters Setting	Number of Test Cases		Average Execution Time (seconds)
	Best	Average	
S1	136	139.6	63.089
S2	118	122.5	1148.0973
S3	113	116.6	4337.3287

Table 4.4 Test case size and execution time in CA(N; 3, 6, 6) configuration

Parameters Setting	Number of Test Cases		Average Execution Time (seconds)
	Best	Average	
S1	387	396.9	215.0552
S2	354	358.8	3270.3555
S3	346	349.3	10784.2423

The first experiment is adopted from (AbdulRahman A Alsewari et al., 2015) which using 10 parameters with 3 values each and the first 30th input-output relationships are utilized. From the result of execution in Table 4.3, S1 setting is generated test cases in the shortest time but it yielded the highest number of test cases generated. S3 has the best number of generated test cases but it consumed very long time to finish the execution. If compared to S2, S3 took approximately four times of S2's time to reduce five test cases to be generated in the best result. It is impractical to consume such a long time to reduce small number of test cases. The number of test cases produced in S1 is reduced significantly compared to S2 which reduced 18 test cases. This result is much more optimum and acceptable to be used.

The second experiment's configuration is originated from (Abdulrahmn A. Alsewari & Zamli, 2011) and the configuration is 6 parameters with 6 values each with uniform interaction strength of 3. The best and average number of test cases generated as well as the average execution time are stated in Table 4.4. S1 in this experiment is still the fastest parameters setting that completed the test case generation. However, the number of test cases it produced is still undesired compared to S2 and S3 settings. The time taken for S3 setting to complete the generation of test cases is approximately 3 hours while S2 only took 55 minutes. The difference of the number of test cases generated

between S3 and S2 is just eight test cases. These issues show that it is unrealistic to use S3 setting in the real environment.

Based on both experiments that are conducted to decide the parameters settings, S2 setting is selected to be parameters setting for all experiments since it is capable to generate the optimum number of test cases in a satisfactory time frame.

4.3.2 Experiments for input-output based relation

There are two input-output based relation experiments conducted as mentioned in section 3.2.4.1. Both experiments adopted the same input-output relationships as stated in Table 3.2. The results of both experiments are presented in Table 4.5 and Table 4.6 respectively.

The result of first experiment which used IOR (N, 3^{10} , R) configuration with the relationships in Table 3.2 is shown in Table 4.5 and the highlighted number of test cases represents the most minimum number of test cases produced out of all strategies. Overall, ITTDG is still outperformed other strategies in term of the size of test cases generated. However, CTJ is still delivered almost optimum solution if compared to the best result generated by ITTDG and ParaOrder. The average difference between the best result of CTJ and other strategies is five test cases only. Besides, the time of execution of CTJ is considerably fast. For R10, CTJ took only approximately 5 minutes to finish the test case generation. 7 minutes, 19 minutes, 27 minutes, 33 minutes and 35 minutes are taken by CTJ to complete the execution of R20, R30, R40, R50 and R60 respectively.

Table 4.6 is the result of execution of experiment two in IOR. The most minimum number of test cases produced by CTJ in R10 and R20 only vary for 7 test cases if compared to Greedy algorithm. While R30 to R60, the difference between the best result of CTJ and Density is not more than 12 test cases. Additionally, the time taken to complete an execution in experiment two is in the range of 500 to 1000 seconds which approximately around 8 to 16 minutes only. These show CTJ generates solutions that are close to optimum.

4.3.3 Experiments for uniform interaction strength

One experiment is carried out to test the efficiency of CTJ in generating test case through uniform interaction strength. There are 14 configurations as mentioned in Table 3.5 are being tested using CTJ and the result of execution is shown in Table 4.7.

Based on the outcome of execution of CTJ, it is observed that most of the strategies including CTJ has generated the most optimal number of test cases except Jenny and TVG. For the remaining configurations, SA and GA are dominant in generating the most optimal number of test cases as they are natural based metaheuristic algorithms. The number of test cases generated through CTJ is still acceptable in overall if compared to the size of test suite produced through exhaustive testing. Furthermore, the results produced by other strategies often go for very high population size and number of iterations for improvement while the parameters setting of CTJ for these experiments are only 500 iterations with the population size of 50. Different parameters setting will affect how well a strategy is performed and hence resulting in different size of test suite produced.

Table 4.5 Test case size and execution time of IOR (N, 3¹⁰, R) configuration in first IOR experiment

R	Density	TVG	ReqOrder	ParaOrder	Union	Greedy	ITTDG	AURA	CTJ		
									Best	Average	Average Execution Time (seconds)
10	86	86	153	105	503	104	81	89	88	90.3	334.6039
20	95	105	148	103	858	110	94	99	100	101.3	444.9273
30	116	125	151	117	1599	122	114	132	118	122.5	1148.0973
40	126	135	160	120	2057	134	122	139	128	130.1	1660.4286
50	135	139	169	148	2635	138	131	147	134	137.8	2006.5269
60	144	150	176	142	3257	143	141	158	145	148.9	2128.8449

Table 4.6 Test case size and execution time of IOR (N, 2³, 3³, 4³, 5¹, R) configuration in second IOR experiment

R	Density	TVG	ReqOrder	ParaOrder	Union	Greedy	ITTDG	AURA	CTJ		
									Best	Average	Average Execution Time (seconds)
10	144	144	154	144	505	137	144	144	144	144.5	509.8875
20	160	161	187	161	929	158	160	182	165	167.1	712.4719
30	165	179	207	179	1861	181	169	200	170	173.2	699.6736
40	165	181	203	183	2244	183	173	207	173	176	748.7497
50	182	194	251	200	2820	198	183	222	191	194.7	842.7382
60	197	209	250	204	3587	207	199	230	209	211.5	987.9181

Table 4.7 Test case size and execution time of different configurations in uniform interaction strength experiment

System Configuration	HSS	SA	GA	ACA	AETG	IPOG	Jenny	TVG	PSTG	CTJ		
										Best	Average	Average Execution Time (seconds)
C1	9	9	9	9	9	9	10	11	9	9	10.7	27.1609
C2	18	16	17	17	17	20	20	19	17	20	20.8	77.2595
C3	155	NA	157	159	NA	176	157	208	NA	182	184	786.1252
C4	341	NA	NA	NA	NA	373	336	473	NA	409	414.8	2198.7327
C5	43	NA	NA	NA	NA	50	45	51	45	46	48.1	172.7624
C6	49	33	33	33	38	53	51	49	42	43	45.6	161.6301
C7	70	64	64	64	77	64	112	123	102	105	108.6	384.8579
C8	199	152	125	125	194	216	215	234	NA	206	209.8	1273.5151
C9	336	300	331	330	330	382	373	407	338	354	358.8	3270.3555
C10	236	201	218	218	218	274	236	271	229	235	239	2152.9612
C11	20	15	15	16	20	19	23	22	NA	22	23	78.6618
C12	48	42	42	42	44	43	50	51	48	50	54	242.4338
C13	119	100	108	106	114	111	131	136	NA	124	127.8	429.332
C14	378	360	360	361	377	383	399	414	385	394	405.2	1969.9214

Note: NA represents Not Available

CHAPTER 5

CONCLUSION

5.1 INTRODUCTION

The aim of this thesis to succeed the fulfilment of CTJ which is an input-output based relation combinatorial testing strategy using Jaya algorithm. The objectives stated in Chapter 1 has attained to realize the aim.

Firstly, conduct a research on existing input-output based relation combinatorial testing strategies which is the first objective have successfully done. This shows in the literature review of Chapter 2 where the introduction of input-output based relation combinatorial testing, all existing uniform interaction strength and IOR combinatorial testing strategies as well as the Jaya algorithm are explained thoroughly.

Moreover, the second objective of this thesis which is implement Jaya algorithm in input-output based relation combinatorial testing has accomplished as well. The methodology and design of CTJ that includes every step of implementation and testing are clearly defined in Chapter 3. There is an example of the execution of CTJ is included in Chapter 4 as well for further understanding the implementation of CTJ.

Furthermore, the evaluation of the performance of CTJ is carried out to realize the last objective of this thesis. Three experiments that includes the evaluation of CTJ in term of performance and efficiency in IOR and uniform interaction strength are carried out to compare with the result of execution of existing combinatorial testing strategies. Based on the result of execution of CTJ in Chapter 4, CTJ is observed perform well especially in test case generation through input-output based relation.

With the successfulness on achieving each objective for this thesis, the aim of this thesis is said to be accomplished. In general, CTJ can generate nearly optimal number of test cases in a considerable time range.

5.2 RESEARCH CONSTRAINTS

There are few constraints that have been found in this research. Firstly, the input parameters and their corresponding values that utilized in the experiments are not the parameters and values from the system under test in the real environment. This is because there is no any real SUT exists that can fulfil all the configuration in both IOR and uniform interaction strength experiments. Hence, a set of mock data is employed to conduct the experiments to get the result of execution.

Moreover, the time taken for CTJ in generating test case for the most complex configuration is found to be long which took around one hour to finish the process. This will waste the time of the tester just to wait for so long to produce the test suite of a particular configuration. Furthermore, lack of customization on the output in CTJ affects the professional software tester to get the specific outcome they desired. Those customizations include disallow the tester to specify certain interaction between values from including or excluding in the test case generation. These constraints should be tackled in the future development of CTJ.

5.3 FUTURE WORKS

From the execution result of CTJ that obtained in Chapter 4, CTJ is found out that there is room for improvement for CTJ. Basically, Jaya algorithm has to be modified in order to improve the performance of CTJ and even generate more optimum number of test cases. The modification may include alteration of the formula of CTJ. In addition, new features can be introduced in CTJ as well to increase the functionality of CTJ. The new features that are suitable for CTJ are adding the support of variable interaction strength, constraints and seeding. Variable interaction strength gives user to set the interaction strength in a more customized way. Further, constraint is a functionality that provides user to restrict certain interactions from being included in test case generation while seeding allows user to define the interaction that must be included in generating the test cases. With the improvement of Jaya algorithm as well as addition of those extra features in the future, CTJ can be become even better and easier to use.

REFERENCES

- Abdul Rahman, A. A. (2012). A harmony search based pairwise sampling strategy for combinatorial testing. *International Journal of the Physical Sciences*, 7(7), 1062 - 1072. doi:10.5897/ijps11.1633
- Abramson, D., & Abela, J. (1991). A parallel genetic algorithm for solving the school timetabling problem.
- Ahmed, B. S., Sahib, M. A., & Potrus, M. Y. (2014). Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing. *Engineering Science and Technology, an International Journal*, 17(4), 218-226. doi:<https://doi.org/10.1016/j.jestch.2014.06.001>
- Ahmed, B. S., & Zamli, K. Z. (2011). A variable strength interaction test suites generation strategy using Particle Swarm Optimization. *Journal of Systems and Software*, 84(12), 2171-2185. doi:<https://doi.org/10.1016/j.jss.2011.06.004>
- Ahmed, B. S., Zamli, K. Z., & Lim, C. P. (2012). Constructing a T-Way Interaction Test Suite Using the Particle Swarm Optimization Approach. *International Journal of Innovative Computing, Information and Control*, 8(1), 431-452.
- Alsewari, A. A., Tairan, N. M., & Zamli, K. Z. (2015). Survey on Input Output Relation based Combination Test Data Generation Strategies. *ARPN Journal of Engineering and Applied Sciences*, 10(18), 8427-8430.
- Alsewari, A. A., & Zamli, K. Z. (2011). *Interaction Test Data Generation Using Harmony Search Algorithm*. Paper presented at the Proceeding of IEEE Symposium on Industrial Electronics & Applications, Langkawi, Malaysia.
- Alsewari, A. R. A., & Zamli, K. Z. (2012). Design and implementation of a harmony-search-based variable-strength t-way testing strategy with constraints support. *Information and Software Technology*, 54(6), 553-568.
- Ammar, M., Bouaziz, S., Alimi, A. M., & Abraham, A. (2013, 12-14 Aug. 2013). *Hybrid harmony search algorithm for global optimization*. Paper presented at the 2013 World Congress on Nature and Biologically Inspired Computing.
- Arshem, J. (2009). TVG. Retrieved from <http://sourceforge.net/projects/tvg>
- Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2(4), 353-373. doi:<https://doi.org/10.1016/j.plrev.2005.10.001>
- Blum, C., & Li, X. (2008). Swarm intelligence in optimization. In *Swarm intelligence* (pp. 43-85): Springer.
- Bryce, R., & Colbourn, C. (2007). *One-Test-at-a-Time Heuristic Search for Interaction Test Suites*. Paper presented at the Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, London, England.
- Bryce, R. C., & Colbourn, C. J. (2009). A Density-Based Greedy Algorithm for Higher Strength Covering Arrays. *Software Testing, Verification & Reliability*, 19(1), 37-53. doi:<http://dx.doi.org/10.1002/stvr.v19:1>
- Chen, X., Gu, Q., Li, A., & Chen, D. (2009, 1-3 Dec. 2009). *Variable Strength Interaction Testing with an Ant Colony System Approach*. Paper presented at the 2009 16th Asia-Pacific Software Engineering Conference.
- Chen, X., Gu, Q., Qi, J., & Chen, D. (2010). *Applying particle swarm optimization to pairwise testing*. Paper presented at the Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual.
- Cohen, D. M., Dalal, S. R., Parelius, J., Patton, G. C., & Bellcore, N. J. (1996). The Combinatorial Design Approach to Automatic Test Generation. *IEEE software*, 13(5), 83-88.

- Cohen, M. B. (2004). *Designing Test Suites for Software Interaction Testing*. (Doctor of Philosophy PhD Thesis), University of Auckland, New Zealand.
- Cohen, M. B., Colbourn, C. J., & Ling, A. C. H. (2003, 17-20 Nov. 2003). *Augmenting simulated annealing to build interaction test suites*. Paper presented at the 14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.
- Cohen, M. B., Gibbons, P. B., Mugridge, W. B., & Colbourn, C. J. (2003). *Constructing Test Suites for Interaction Testing*. Paper presented at the Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon USA.
- Colbourn, C. J., Cohen, M. B., & Turban, R. (2004). *A deterministic density algorithm for pairwise interaction coverage*. Paper presented at the IASTED Conf. on Software Engineering.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms third edition. *MIT Press. ISBN 0-262-03384-4. Section, 23*, 631-638.
- De Vries, S., Vohra, R., Economics, N. U. C. f. M. S. i., & Science, M. (2003). Combinatorial auctions: A survey. *INFORMS Journal on Computing*, 15(3), 284-309.
- Eberhart, R., & Kennedy, J. (1995). *A new optimizer using particle swarm theory*. Paper presented at the Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on.
- Elbeltagi, E., Hegazy, T., & Grierson, D. (2005). Comparison among five evolutionary-based optimization algorithms. *Advanced Engineering Informatics*, 19(1), 43-53. doi:<https://doi.org/10.1016/j.aei.2005.01.004>
- Geem, Z. W., & Kim, J. H. (2001). A New Heuristic Optimization Algorithm: Harmony Search. *Simulation*, 76(2), 60-68.
- Geem, Z. W., Kim, J. H., & Loganathan, G. V. (2001). A New Heuristic Optimization Algorithm: Harmony Search. *Simulation*, 76(2), 60-68. doi:10.1177/003754970107600201
- Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*: MIT press.
- Ingber, L. (1993). Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling*, 18(11), 29-57. doi:[https://doi.org/10.1016/0895-7177\(93\)90204-C](https://doi.org/10.1016/0895-7177(93)90204-C)
- Jenkins, B. (2003, February 5 2005). Jenny. Retrieved from <http://burtleburtle.net/bob/math/jenny.html>
- Kennedy, J., & Eberhart, R. (1995). *Particle Swarm Optimization*. Paper presented at the Proceedings of IEEE International Conference Neural Networks.
- Khazali, A. H., & Kalantar, M. (2011). Optimal reactive power dispatch based on harmony search algorithm. *International Journal of Electrical Power & Energy Systems*, 33(3), 684-692. doi:<https://doi.org/10.1016/j.ijepes.2010.11.018>
- Kuhn, D. R., Wallace, D. R., & Gallo, A. M. (2004). Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering*, 30(6), 418-421. doi:10.1109/TSE.2004.24
- Lam, S. S. B., Raju, M. L. H. P., M, U. K., Ch, S., & Srivastav, P. R. (2012). Automated Generation of Independent Paths and Test Suite Optimization Using Artificial Bee Colony. *Procedia Engineering*, 30, 191-200. doi:<https://doi.org/10.1016/j.proeng.2012.01.851>
- Lei, Y., Kacker, R., Kuhn, D. R., Okun, V., & Lawrence, J. (2007). *IPOG: A General Strategy for T-Way Software Testing*. Paper presented at the Proceedings of the

- 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, Tucson, AZ U.S.A.
- Liang, X., Guo, S., Huang, M., & Jiao, X. (2014). Combinatorial Test Case Suite Generation Based on Differential Evolution Algorithm. *JSW*, 9(6), 1479-1484.
- Mao, C., Yu, X., Chen, J., & Chen, J. (2012, 27-29 Aug. 2012). *Generating Test Data for Structural Testing Based on Ant Colony Optimization*. Paper presented at the 2012 12th International Conference on Quality Software.
- McCaffrey, J. D. (2009, 20-24 July 2009). *Generation of Pairwise Test Sets Using a Genetic Algorithm*. Paper presented at the Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference.
- McCall, J. (2005). Genetic algorithms for modelling and optimisation. *Journal of Computational and Applied Mathematics*, 184(1), 205-222.
doi:<https://doi.org/10.1016/j.cam.2004.07.034>
- Mishra, S., & Ray, P. K. (2016). Power quality improvement using photovoltaic fed DSTATCOM based on JAYA optimization. *IEEE Transactions on Sustainable Energy*, 7(4), 1672-1680.
- Ong, H. Y., & Zamli, K. Z. (2011). Development of Interaction Test Suite Generation Strategy with Input-Output Mapping Supports. *Scientific Research and Essays*, 6(16), 3418-3430. doi:Available online at <http://www.academicjournals.org/SRE>
- Othman, R. R., & Zamli, K. Z. (2011). ITTDG: Integrated T-way Test Data Generation Strategy for Interaction Testing. *Scientific Research and Essays*, 6(17), 3638-3648. doi:Available online at <http://www.academicjournals.org/SRE>
- Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization. *Swarm intelligence*, 1(1), 33-57.
- Ramli, N., Othman, R. R., & Ali, M. S. A. R. (2016, 11-12 Aug. 2016). *Optimizing combinatorial input-output based relations testing using Ant Colony algorithm*. Paper presented at the 2016 3rd International Conference on Electronic Design (ICED).
- Rao, R. (2016). Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *International Journal of Industrial Engineering Computations*, 7(1), 19-34.
- Rao, R., More, K., Taler, J., & Ocloń, P. (2016). Dimensional optimization of a micro-channel heat sink using Jaya algorithm. *Applied Thermal Engineering*, 103, 572-582.
- Rao, R. V., Savsani, V. J., & Vakharia, D. (2011). Teaching-learning-based optimization: a novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, 43(3), 303-315.
- Schroeder, P. J. (2001). *Black-Box Test Reduction Using Input-Output Analysis*. (Ph.D. Ph.D.), Illinois Institute of Technology.,
- Schroeder, P. J., Faherty, P., & Korel, B. (2002, 2002). *Generating expected results for automated black-box testing*. Paper presented at the Proceedings 17th IEEE International Conference on Automated Software Engineering.
- Schroeder, P. J., & Korel, B. (2000). Black-box test reduction using input-output analysis. *SIGSOFT Softw. Eng. Notes*, 25(5), 173-177.
doi:<http://doi.acm.org/10.1145/347636.349042>
- Selvi, V., & Umarani, D. R. (2010). Comparative analysis of ant colony and particle swarm optimization techniques. *International Journal of Computer Applications* (0975-8887), 5(4).

- Shiba, T., Tsuchiya, T., & Kikuno, T. (2004a, 28-30 Sept. 2004). *Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing*. Paper presented at the Proceedings of the 28th Annual International Computer Software and Applications Conference.
- Shiba, T., Tsuchiya, T., & Kikuno, T. (2004b, 28-30 Sept. 2004). *Using artificial life techniques to generate test cases for combinatorial testing*. Paper presented at the Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.
- Shin, K.-S., & Lee, Y.-J. (2002). A genetic algorithm application in bankruptcy prediction modeling. *Expert Systems with Applications*, 23(3), 321-328. doi:[https://doi.org/10.1016/S0957-4174\(02\)00051-9](https://doi.org/10.1016/S0957-4174(02)00051-9)
- Singh, S. P., Prakash, T., Singh, V., & Babu, M. G. (2017). Analytic hierarchy process based automatic generation control of multi-area interconnected power system using Jaya algorithm. *Engineering Applications of Artificial Intelligence*, 60, 35-44.
- Srivastava, P. R., & Kim, T.-h. (2009). Application of genetic algorithm in software testing. *International Journal of software Engineering and its Applications*, 3(4), 87-96.
- Stardom, J. (2001). *Metaheuristics and the search for covering and packing arrays*: Simon Fraser University.
- Tricentis. (2018). *Software Fail Watch: 5th Edition*. Retrieved from https://www.tricentis.com/wp-content/uploads/2018/02/20180207_Software-Fails-Watch.pdf
- Vesterstrom, J., & Thomsen, R. (2004). *A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems*. Paper presented at the IEEE Congress on Evolutionary Computation.
- Wang, Z., Xu, B., & Nie, C. (2008, 12-13 Aug. 2008). *Greedy Heuristic Algorithms to Generate Variable Strength Combinatorial Test Suite*. Paper presented at the 2008 The Eighth International Conference on Quality Software.
- Wang, Z. Y., Xu, B. W., & Nie, C. H. (2008). *Greedy Heuristic Algorithms to Generate Variable Strength Combinatorial Test Suite*. Paper presented at the Proceedings of the 8th International Conference on Quality Software.
- Warid, W., Hizam, H., Mariun, N., & Abdul-Wahab, N. I. (2016). Optimal power flow using the Jaya algorithm. *Energies*, 9(9), 678.
- Wu, H., Nie, C., Kuo, F. C., Leung, H., & Colbourn, C. J. (2015). A Discrete Particle Swarm Optimization for Covering Array Generation. *IEEE Transactions on Evolutionary Computation*, 19(4), 575-591. doi:10.1109/TEVC.2014.2362532
- Xambre, A. R., & Vilarinho, P. M. (2003). A simulated annealing approach for manufacturing cell formation with multiple identical machines. *European journal of operational research*, 151(2), 434-446.
- Xiang, L. Y., Alsewari, A. A., & Zamli, K. Z. (2015). Pairwise test suite generator tool based on harmony search algorithm (HS-PTSGT). *International Journal on Artificial Intelligence*, 2.
- Yu-Wen, T., & Aldiwan, W. S. (2000). *Automating Test Case Generation for the New Generation Mission Software System*. Paper presented at the Proceedings of the IEEE Aerospace Conference, Big Sky, MT, USA.
- Ziyuan, W., Changhai, N., & Baowen, X. (2007). *Generating combinatorial test suite for interaction relationship*. Paper presented at the Proceeding of the 4th

international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting, Dubrovnik, Croatia.

APPENDIX A GANTT CHART

