DESIGN AND IMPLEMENTATION OF A T-WAY TUPLE TREE GENERATOR FOR T-WAY TEST CASE GENERATION STRATEGY

By

Rafsan Ahmed Al-Rafi
ID: 151-35-1037

This Report Presented in Partial Fulfilment of the Requirements for the Degree of Bachelor of Science in Software Engineering

DEPARTMENT OF SOFTWARE ENGINEERING
DAFFODIL INTERNATIONAL UNIVERSITY

FALL 2018

APPROVAL

This thesis titled on **“Design and Implementation of a T-way Tuple Tree Generator for T-way Test Case Generation Strategy’’**, submitted by **Rafsan Ahmed Al-Rafi, 151-35-1037,** to the department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

THESIS DECLARATION

I hereby declare that, this Thesis Report has been done by me under the supervision of Dr. Md. Mostafijur Rahman, Assistant Professor, Department of Software Engineering (SWE), Faculty of Science and Information Technology (FSIT), Daffodil International University (DIU). I also declare that, neither this report nor any part of this report has been submitted elsewhere for award of any degree.

Submitted By

…………………………………………………….

Rafsan Ahmed Al-Rafi

ID: 151-35-1037

Department of Software Engineering

Faculty of Science and Information Technology

Daffodil International University

Certified By

[illegible] 22/12/2018

……………………………………………………

Dr. Md. Mostafijur Rahman

Assistant Professor

Department of Software Engineering

Faculty of Science and Information Technology

Daffodil International University

ACKNOWLEDGEMENT

Alhamdulillah, all praises to the Almighty Allah who gives me the ability, benediction, motivation, patience and wisdom to complete this research work.

I would like to extend my gratitude to my respectful supervisor Dr. Md. Mostafijur Rahman. Without him and his excellent guidance, motivation, care, patience, and providing me with excellent facilities and environment for doing this research, this Research and Thesis would not exist. I am also grateful to my parents and the people around me who always keep supporting me.

TABLE OF CONTENTS

PAGE

APPROVAL ………………………………………………………………………........ i

THESIS DECLARATION ……………………………………………………………….. ii

ACKNOWLEDGEMENT ……………………………………………………………….. iii

TABLE OF CONTENTS ………………………………………………………………. iv

LIST OF FIGURES ……………………………………………………………………. vi

LIST OF TABLES ……………………………………………………………………... vii

LIST OF ABBREVIATIONS …………………………………………………………. vii

ABSTRACT ……………………………………………………………………………. ix

CHAPTER 1: INTRODUCTION ……………………………………………………… 1

1.1 Background ………………………………………………………………………… 1

1.2 Motivation of the Research ……………………………………………………….... 1

1.3 Problem Statement …………………………………………………………………. 2

1.4 Research Question …………………………………………………………………. 3

1.5 Research Objectives ……………………………………………………………….... 3

1.6 Research Scope ……………………………………………………………………... 4

1.7 Thesis Organization ………………………………………………………………… 4

CHAPTER 2: LITERATURE REVIEW ……………………………………………… 5

2.1 Preliminaries ……………………………………………………………………….. 5

2.2 Existing t-way Strategies for Sequence-less Input Interaction …………………… 5

2.2.1 High Level Hyper Heuristic (HHH) ..……………………………………… 5

2.2.2 Harmony Search Strategy (HSS) ………...……………………………….. 5

2.2.3 Particle Swarm based Test Generator (PSTG)…………………………. 5

2.2.4 Cuckoo Search Strategy (CSS)……………………………………….. 6

2.2.5 Simulated Annealing (SA)……………………………………………. 6

2.2.6 Genetic Algorithm (GA) ……………………………………………... 6

2.2.7 Ant Colony Algorithm (ACA) ………………………………………. 6

2.2.8 Bat-Inspired T-way Strategy (BTS) …………………………………. 6

2.2.9 Late Acceptance Hill Climbing (LAHC) ……………………………. 7

2.2. 10 (GA-N) ……………………………………………………………… 7

2.3 Analysis and discussion on existing strategies ……………………………..... 7

2.4 Summary ………………………………………………………………….. 7

CHAPTER 3: RESEARCH METHODOLOGY …………………………………... 9

3.1 Sequence-less Combinatorial Input Interaction Testing (CIIT) ……………….. 9

3.2 A test scenario ......……………………………………………………….. 13

3.2 The test scenario formed as a problem …….……………………………….. 14

3.3 Exhaustive test cases for the scenario ……………………………………….. 14

3.4 Representing the scenario as a tree ……………………………………….. 15

3.5 Narrowing down the tree …………………………………………………. 15

3.6 Turning the scenario into a matrix ……………………………………….. 16

3.7 Breaking down the matrix ……………………………………………….. 16

3.8 Tuples creation process (for T-ways, where T=2) ……………………………. 17

3.9 Enhanced T-way Tuple Tree Generator Algorithm (ETTTG Algorithm) ……. 20

3.10 Summary ………………………………………………………………… 20

CHAPTER 4: EXPERIMENTAL RESULTS AND DISCUSSION …………….. 21

4.1 Input-output for the scenario of the research ………………………………… 21

4.2 Comparison with Generic Tuple Generator and benchmarking ……………... 22

4.3 Discussion of the result …………………………………………………….. 23

4.3 Summary ………………………………………………………………… 23

CHAPTER 5: CONCLUSION AND FUTURE WORKS ……………………....... 24

5.1 Findings and Contributions ………………………………………………… 24

5.2 Limitation ………………………………………………………………... 24

5.3 Recommendations for Future Works ……………………………………… 25

REFERENCES …………………………………………………………….... 26

LIST OF FIGURES

NO. PAGE

1.1 How a system should work ……………………………………………….. 2

1.2 How a system may fail because of problems in inputs ………………………... 2

1.3 Limitations of existing solutions ……………………………………………… 3

1.4 Scope space of this research …………..……………………………………… 4

3.1 Framework for sequence-less CIIT ……….…………………………… …. 9

3.2 An example of sequence-less CIIT for uniform values ………………………. 10

3.3 An example of sequence-less CIIT for non-uniform values …………………. 12

3.4 A test scenario ……………………………………………………………….. 13

3.5 The test scenario formed as a problem …..……………………..……………. 14

3.6 Exhaustive test cases ………………………………..……………………. 14

3.7 Tree representation of the scenario ………………………………………… 15

3.8 The narrowed down tree ……………………………………………………. 15

3.9 Matrix representation of the scenario ………………………………………. 16

3.10 Break-down of the matrix from Fig 3.9 ……………………………………. 16

3.11 Assigning values from Fig 3.10 …………………………………………….. 17

3.12 Tuple creation process for Column 0 (Input A) ……………………………. 17

3.13 Summary of the entire tuple making process ………………………………… 18

3.14 ETTTG Algorithm ………………………………………………………………. 20

4.1 Output for the scenario of the research ……………………………………….. 22

LIST OF TABLES

NO. **PAGE**

2.1 Algorithms/tools published by year ………………………………………. 8

3.1 Generated exhaustive test cases for Fig 3.2 ………………………………. 10

3.2 3-way tuples for the uniform values ………………………………………. 11

3.3 Generated exhaustive test cases for Fig 3.3 ………………………………. 12

4.1 Comparison between GTG and ETTTG and Benchmarking ……………… 22

LIST OF ABBREVIATIONS

CIIT	Combinatorial Input Interaction Testing
CA	Covering Array
GTG	Generic Tuple Generator

ETTTG	Enhanced T-way Tuple Tree Generator
ISTQB	International Software Testing Qualification Board
QA	Quality Assurance
NP-hard	Non-deterministic Polynomial-time Hard
ACA	Ant Colony Algorithm
SA	Simulated Annealing
PSO	Particle Swarm Optimization
PSTG	Particle Swarm Test Suite Generator
BTS	Bat-inspired Test Strategy
HHH	High-level Hyper Heuristic
CSA	Cuckoo Search Algorithm

ABSTRACT

Combinatorial Input Interaction Testing (CIIT) can either be sequence based or sequence less for either uniform or non-uniform values. It is an NP-hard problem, because there is no exact solution for this problem that has the best result. Thus, there are many solutions for this problem. Some of the solutions are based on Integrated T-way Test Suite Generator (ITTSG), Generalized T-way Test Suite Generator (GTWay), Particle Swarm Algorithm, Harmony Search Algorithm, Hill Climbing Algorithm, Ant Colony Algorithm, High-level Hyper Heuristic (HHH) Algorithm, Meta-Heuristic Algorithm, Bat-inspired Algorithm etc. Few solutions are based on T-way test case generation strategy. The methodology of this research is a T-way tuple tree generation algorithm for T-way test case generation strategies. This algorithm is named “Enhanced T-way Tuple Tree Generator”. This algorithm accomplishes a faster and more efficient tuple tree generation process; faster because it takes less time to generate tuple trees and efficient because it eliminates all kind of redundancy and allows more inputs to work with.

CHAPTER 1

INTRODUCTION

1.1 Background

Modern society is being built upon software and this is an ever-increasing scenario. The more the world advances, the more it will depend on software. Software will control and maintain many aspect of our lives. The current impact of software on modern society is huge. We use software each and every day in our life.

Thus, software should be tested for security and expected work assurance. Software Testing is an integral part of Software Development Life Cycle (SDLC).

A software can have many inputs and their combinations and this is called Combinatorial Input Interaction. These combinations of inputs should also be tested for Quality Assurance (QA) of a software. This is called Combinatorial Input Interaction Testing (CIIT).

CIIT can be sequence less or sequence based. Both sequence less and sequence based CIIT can also be for uniform or non-uniform values.

This research deals with sequence less CIIT for both uniform and non-uniform values by using an Enhanced T-way Tuple Tree Generator Algorithm (ETTTG Algorithm).

1.2 Motivation of the Research

There are so many existing solutions on CIIT that deals with combinatorial input interaction optimization problems in different ways. But very few of them focuses on T-way tuple trees and those that focuses on the T-way tuple generation, they do not really deal with optimizing the tuple generation process. Thus, the main focus is on optimizing the tuple tree generation process and making it faster and more efficient so that the entire process can be faster and more efficient.

1.3 Problem Statement

Combinatorial Input Interaction Testing (CIIT) is an area of software testing that deals with the combinations of inputs and interactions between them. Testing is a crucial part of software development and CIIT is a crucial part of software testing and quality assurance.

The following figure shows how a system should work:

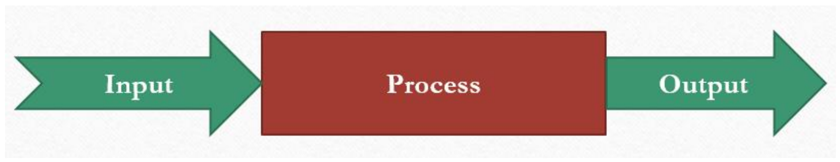


Fig 1.1: How a system should work

And the following figure shows what happens if there is problem in inputs:

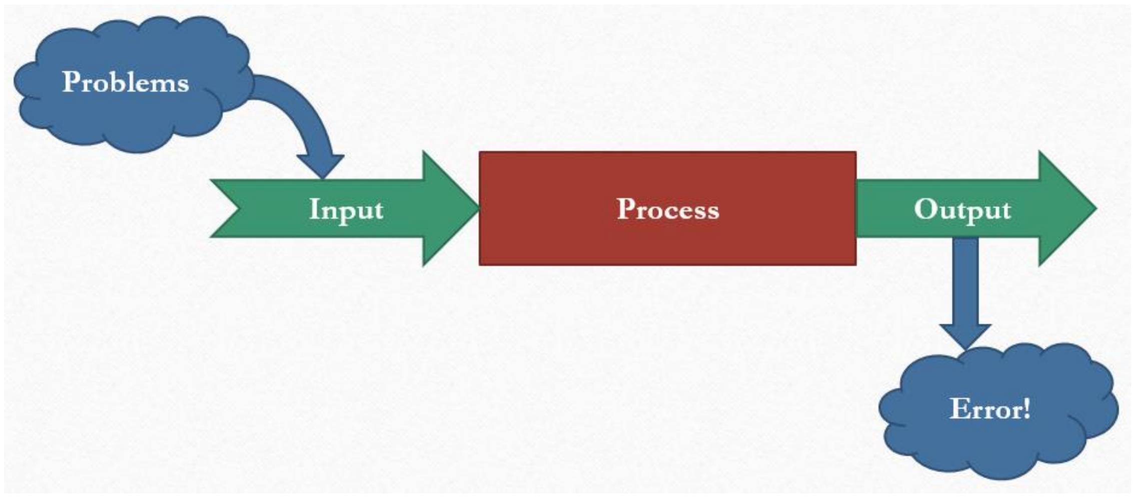
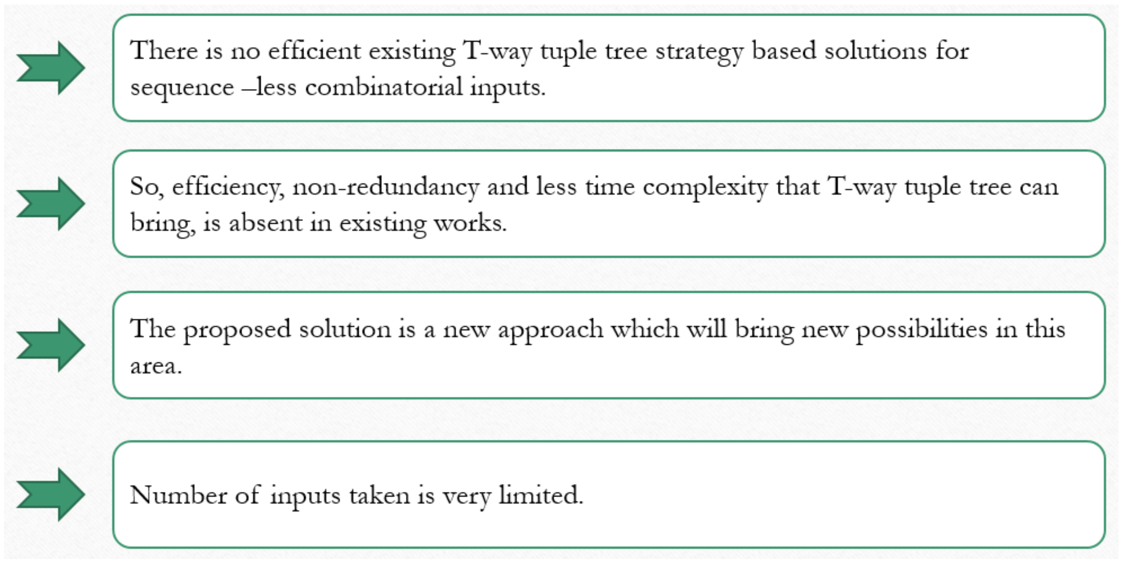


Fig 1.2: How a system may fail because of problems in inputs

Therefore, CIIT is a used to ensure interaction of inputs are done correctly.

There are many existing strategies to deal with CIIT. Some of the strategies include creation of T-way tuple trees. But the existing solutions have some limitations.

These are the limitations of the existing solutions:


There is no efficient existing T-way tuple tree strategy based solutions for sequence –less combinatorial inputs.

So, efficiency, non-redundancy and less time complexity that T-way tuple tree can bring, is absent in existing works.

The proposed solution is a new approach which will bring new possibilities in this area.

Number of inputs taken is very limited.

Fig 1.3: Limitations of existing solutions

Therefore, the goal of my thesis is to work on sequence-less CIIT based on T-way tuple tree strategy to overcome these limitations. And creating an enhanced T-way tuple tree generator for T-way test case generation strategies to deal with CIIT is the main focus point of this thesis.

1.4 Research Question

Does this t-way tuple tree generation algorithm makes the tuple tree generation process faster and allow more inputs to work with and eliminate redundancy?

1.5 Research Objectives

The aim of this research is to design and implement a T-way tuple tree generator for T-way test case generation strategy. To achieve this aim, the following objectives are taken under consideration:

1. To design and implement a T-way tuple tree generator for T-way test case generation strategy.
2. To minimize time complexity and to increase efficiency and number of inputs taken.
3. To compare with other existing tuple generator.

1.6 Research Scope

The scope of this thesis is to create an enhanced t-way tuple tree generator for t-way test case generation strategies. The following figure shows the research scope of this thesis:

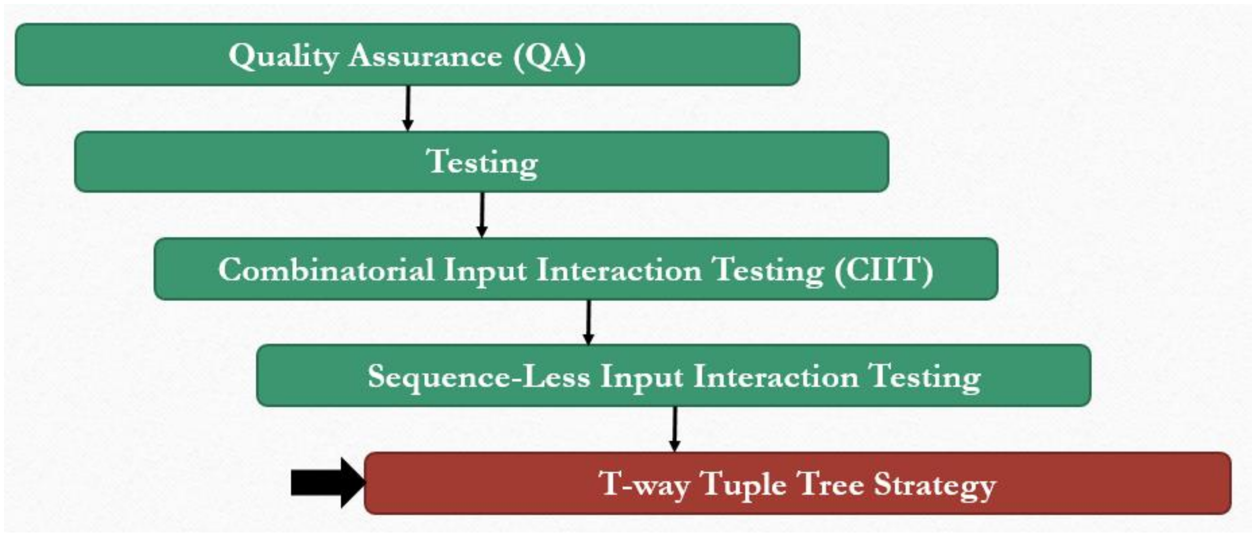

Quality Assurance (QA)
Testing
Combinatorial Input Interaction Testing (CIIT)
Sequence-Less Input Interaction Testing
T-way Tuple Tree Strategy

Fig 1.4: Scope space of this research

1.7 Thesis Organization

Chapter 1 contains background, motivation, problem statement, objectives and scope of the research.

Chapter 2 contains literature review. And literature review contains definitions and discussions related to the research topic and also existing solutions.

Chapter 3 presents, discusses and visualizes the design, algorithm and implementation of a new strategy for sequence-less CIIT based on T-way tuple tree generation strategy.

Chapter 4 contains the evaluation of the tuple tree generation algorithm and comparison with other existing solutions.

Finally, Chapter 5 summarizes the achievements and limitations of the proposed design, algorithm and implementation.

Chapter 5 concludes this research work with some recommendations for future work.

CHAPTER 2

LITERATURE REVIW

This chapter contains discussion and analysis on sequence-less CIIT for uniform and non-uniform values, the existing solutions and the proposed solution.

2.1 Preliminaries

CIIT can be both sequence based or sequence-less for both uniform and non-uniform values. There are so many existing solutions that deal with CIIT using different strategies and algorithms. Some of the existing solutions use t-way tuple tree generation strategy for sequence-less combinatorial input interaction.

2.2 Existing t-way Strategies for Sequence-less Input Interaction

In this section, a number of existing t-way strategies have been discussed that have been developed and improved throughout the years.

2.2.1 High Level Hyper Heuristic (HHH)

The High Level Hyper Heuristic (HHH) (Zamli et al., 2017) strategy is a hybrid t-way test case generation strategy. The selection and acceptance of the four low level meta-heuristic algorithms is based on the improvement, diversification and intensification operator (like other hyper met heuristic algorithms). In HHH, the adopted low level meta-heuristic algorithms are designed for continuous problems.

2.2.2 Harmony Search Strategy (HSS)

Alsewari and Zamli (2011), adopted harmony search (HS) meta-heuristic algorithm for t-way test strategy to generate test suite. It is population-based algorithm. The HS uses a probabilistic-gradient in its search space and to select the current solution to adopt mathematical equations for better solution. It is proved that the harmony search algorithm perform well in solving highly interactive combinatorial problems (Alsewari and Zamli, 2011).

2.2.3 Particle Swarm based Test Generator (PSTG)

Ahmed et al. (2012a; 2012b) designed a t-way test suite generation strategy called particle swarm test generator (PSTG). It is adopted by particle swarm optimization (PSO) (Ahmed et al., 2012a; Ahmed et al., 2012b; Mahmoud & Ahmed, 2015).It is also population based optimization method (Kennedy & Eberhart, 1995a; Kennedy & Eberhart, 1995b).PSO comprises a group of particles with negligible mass and volume and which move through hyperspace**.**

2.2.4 Cuckoo Search Strategy (CSS)

Cuckoo search strategy (CSS) (Nasser et al., 2015) is a recent strategy for t-way test generation. It generates random initial nests. Each egg in a nest represents a vector solution indicates a test case. Firstly, a new nest is generated through levy flight path (Yang & Deb, 2009). Then it is evaluated against the existing nests. If there is found a better result, the new nest is replaced as current nest. Secondly, CS has probabilistic elitism in order to maintain elite solutions for the next generation.

2.2.5 Simulated Annealing (SA)

Cohen et al. (Cohen et al., 2003a) used simulated annealing (SA) to solve t-way combinatorial problem. This is also a heuristic searching method to achieve optimal test suite. In this technique, an initial feasible solution is set as a best solution then compare with the best solution. A transformation function is used to select the next feasible solution. The cooling rate and temperature are used to control the iteration (Cohen et al., 2003b).

2.2.6 Genetic Algorithm (GA)

Genetic algorithm (GA) proposed for t-way test strategy to generate test suite (Shiba et al., 2004). It is the process of natural selection. It begins with randomly created test cases, based on chromosomes. These crossover and mutation is happening until a termination criterion is met. The goodness of a candidate function estimate by use fitness function. A selection function selects a number of good candidate solution. The best chromosomes are selected and added to the final test suite.

2.2.7 Ant Colony Algorithm (ACA)

Ant colony optimization (ACO) algorithm adopted on t-way test strategy (Shiba et al., 2004). It is the behavior of natural ant colonies to find paths from the colony to food. The candidate solutions are determined by each path from a starting point to an ending point that associated with the candidate solution. The amount of pheromone deposited in each ant movement path are selected based on the candidate solution the next candidate solution is based on the larger number of pheromone. Finally, there may have possibility to achieve near optimum or optimum solution to the target problem.

2.2.8 Bat-Inspired T-way Strategy (BTS)

Alsariera & Zamli (2015), bat algorithm was adapted for t-way strategy to generate test suite, which is called bat-inspired testing strategy (BTS). The bat algorithm (BA) (Yang, 2010) is a natural-inspired algorithm. The interpretation of the nature may not perfect. The BA is a population optimization algorithm. The Bats find its best moving dimension from its position and velocity. In every iteration, the bat algorithm provides an exhaustive local search method throughout its random walk behavior to find the best solution.

2.2.9 Late Acceptance Hill Climbing (LAHC)

Late Acceptance based Hill Climbing (LAHC) is a heuristic search algorithm (Zamli et al., 2015). When a candidate cost function is better (or equal) which accept non improving moves. Each current solution is employed during the later (not immediate) acceptance procedure. LAHC is started from a randomly generated initial solution and it evaluates a new candidate in order to accept or reject at each iteration. The last element is compared with the candidate cost of the list and if not worse than accepted. After the acceptance procedure, the cost of the new current solution is inserted into the beginning of the list and the last element is removed from the end of the list. When the inserted current cost is equal to the candidate's cost in the case of accepting only, but in the case of rejecting it is equal to the previous value (Burke & Bykov, 2017).

2.2.10 (GA-N)

The GA-N is the upgraded version of GA (Shiba et al., 2004). where N indicates N (N = 2, 3, 4, 5, 6…) number of interactions.

2.3 Analysis and discussion on existing strategies

Most of the existing solutions work on covering array (CA) and mainly focuses on minimizing the exhaustive test cases. But none of the solutions deal with making the tuple tree generation faster and more efficient remarkably.

For this reason, my solution is purely based on making the unique tuples generation process faster and more efficient significantly.

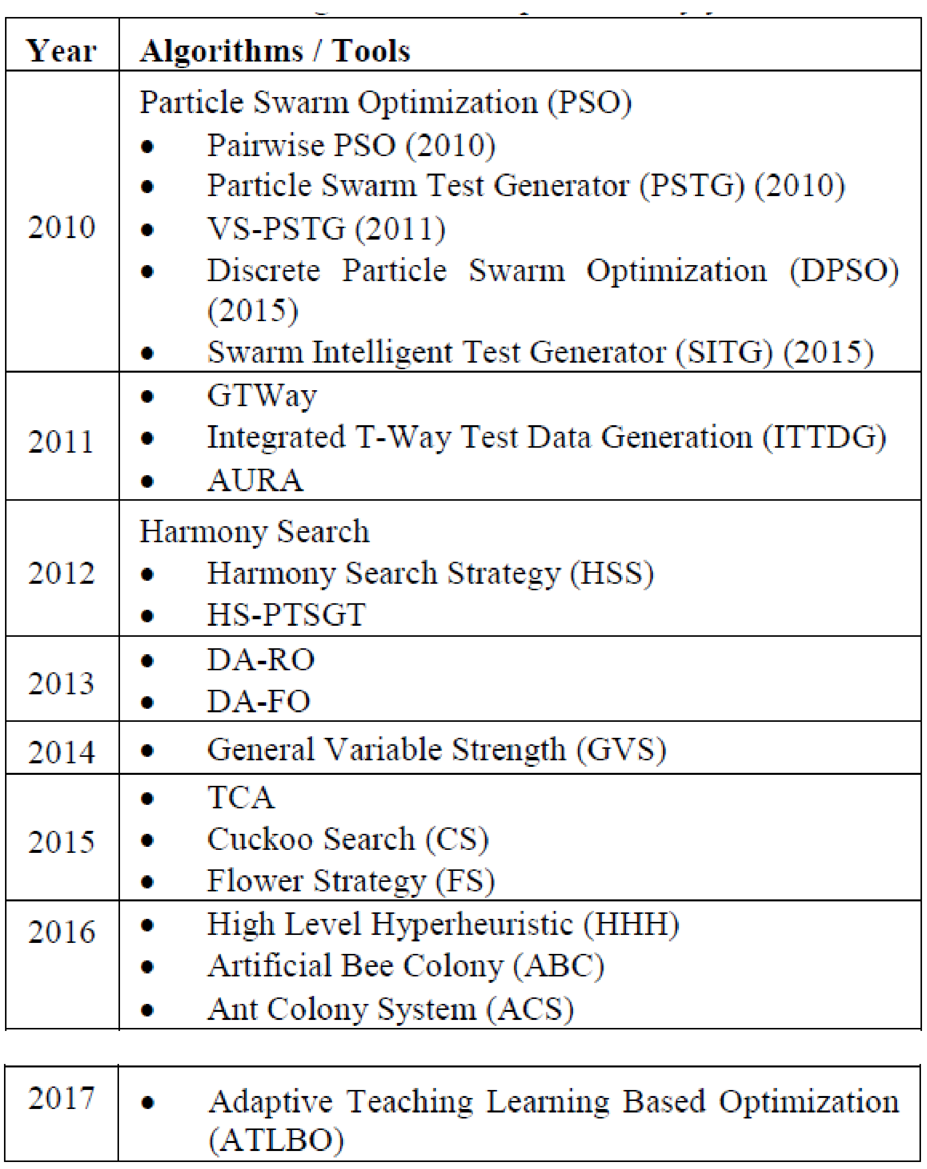
2.4 Summary

There are so many existing solutions to deal with sequence-less CIIT. Some of the solutions are based on t-way test case generation strategy. T-way test case generation strategies require tuple tree generation.

My thesis only focuses on tuple tree generation process and doesn't deal with minimizing exhaustive test cases, which is the gap of my thesis and can be worked on in the future.

The following table shows a summary of existing t-way strategies:

Table 2.1 Algorithms/tools published by year (Nuraminah Ramli et al., 2017)



Year	Algorithms / Tools
2010	Particle Swarm Optimization (PSO) • Pairwise PSO (2010) • Particle Swarm Test Generator (PSTG) (2010) • VS-PSTG (2011) • Discrete Particle Swarm Optimization (DPSO) (2015) • Swarm Intelligent Test Generator (SITG) (2015)
2011	• GTWay • Integrated T-Way Test Data Generation (ITTDG) • AURA
2012	Harmony Search • Harmony Search Strategy (HSS) • HS-PTSGT
2013	• DA-RO • DA-FO
2014	• General Variable Strength (GVS)
2015	• TCA • Cuckoo Search (CS) • Flower Strategy (FS)
2016	• High Level Hyperheuristic (HHH) • Artificial Bee Colony (ABC) • Ant Colony System (ACS)
2017	• Adaptive Teaching Learning Based Optimization (ATLBO)

CHAPTER 3

RESEARCH METHODOLOGY

Combinatorial Input Interaction Testing has two parts. Sequence-based CIIT and sequence-less CIIT. Both sequence-based and sequence-less CIIT also may deal with uniform or non-uniform values. And these are NP-hard problem because they don't have any actual and universal optimum solution.

The following part discusses about sequence-less CIIT:

3.1 Sequence-less Combinatorial Input Interaction Testing (CIIT)

The following figure shows the framework for sequence-less CIIT:

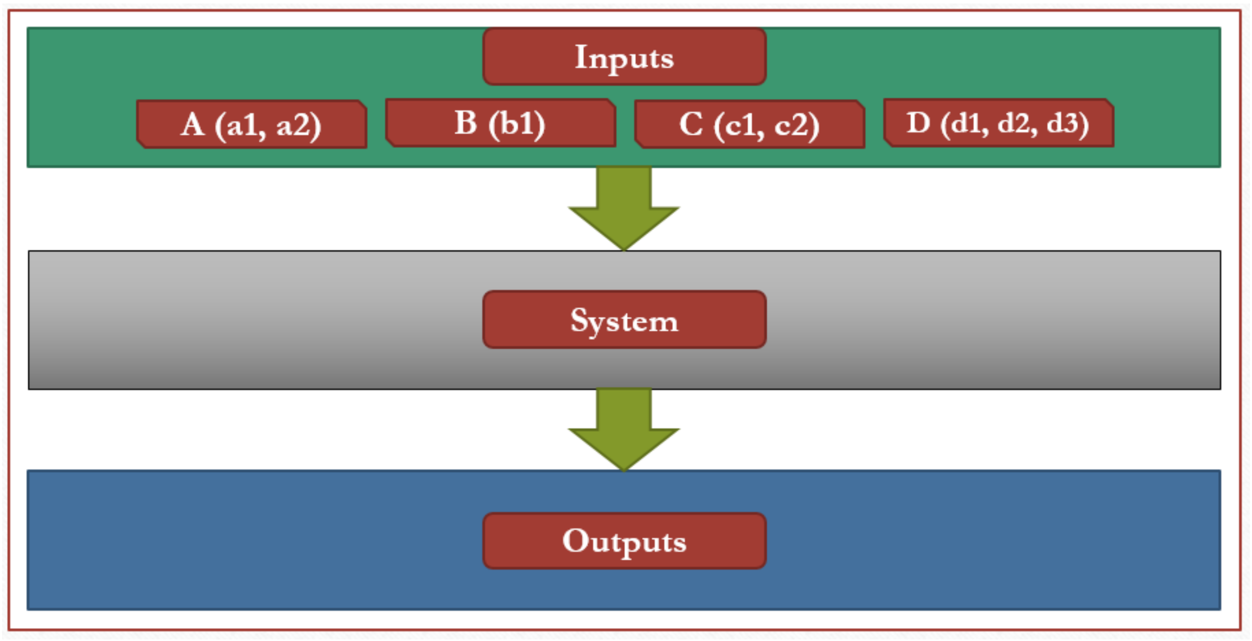


Fig 3.1: Framework for sequence-less CIIT

A. **Sequence-less input interaction with uniform values:**

The following figure shows an example of sequence-less CIIT for uniform values:

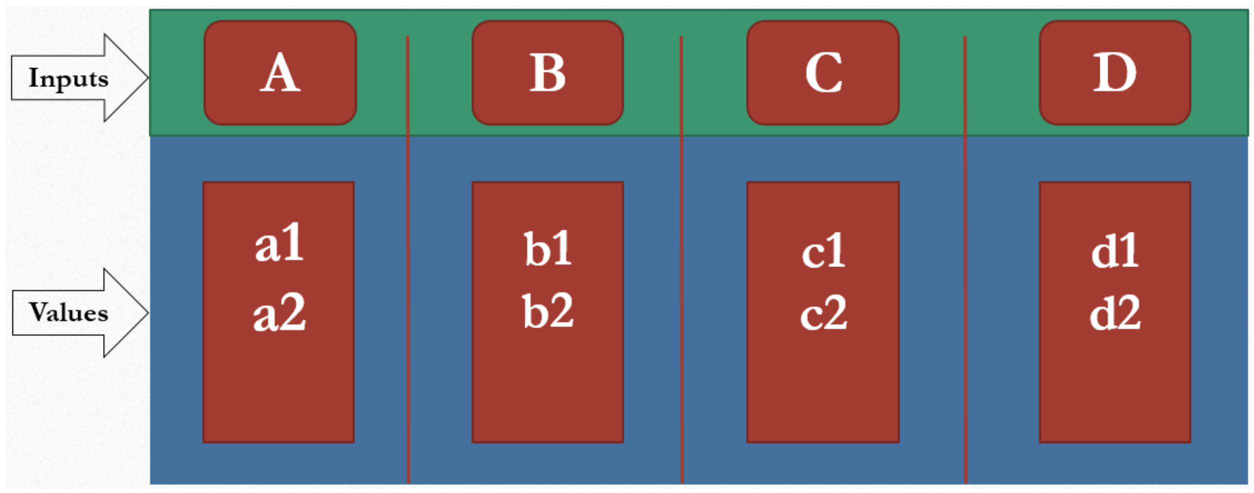


Fig 3.2: An example of sequence-less CIIT for uniform values.

The number of exhaustive test cases should be 2*2*2*2 = 16.

Now, the exhaustive test cases for this example are:

Table 3.1: Generated exhaustive test cases for **Fig 3.2**

No.	**Exhaustive test case**
1	[a1,b1,c1,d1]
2	[a1,b1,c1,d2]
3	[a1,b2,c1,d1]
4	[a1,b2,c1,d2]
5	[a1,b1,c2,d1]
6	[a1,b1,c2,d2]
7	[a1,b2,c2,d1]
8	[a1,b2,c2,d2]
9	[a2,b1,c1,d1]

10	[a2,b1,c1,d2]
11	[a2,b2,c1,d1]
12	[a2,b2,c1,d2]
13	[a2,b1,c2,d1]
14	[a2,b1,c2,d2]
15	[a2,b2,c2,d1]
16	[a2,b2,c2,d2]

Table 3.2: 3-way tuples for the uniform values

No	**3-way tuple**	**No**	**3-way tuple**	**No**	**3-way tuple**	**No**	**3-way tuple**
1	[a1, b1, c1]	9	[a1, b1, d1]	17	[a1, c1, d1]	25	[b1, c1, d1]
2	[a1, b1, c2]	10	[a1, b1, d2]	18	[a1, c1, d2]	26	[b1, c1, d2]
3	[a1, b2, c1]	11	[a1, b2, d1]	19	[a1, c2, d1]	27	[b1, c2, d1]
4	[a1, b2, c2]	12	[a1, b2, d2]	20	[a1, c2, d2]	28	[b1, c2, d2]
5	[a2, b1, c1]	13	[a2, b1, d1]	21	[a2, c1, d1]	29	[b2, c1, d1]
6	[a2, b1, c2]	14	[a2, b1, d2]	22	[a2, c1, d2]	30	[b2, c1, d2]
7	[a2, b2, c1]	15	[a2, b2, d1]	23	[a2, c2, d1]	31	[b2, c2, d1]
8	[a2, b2, c2]	16	[a2, b2, d2]	24	[a2, c2, d2]	32	[b2, c2, d2]

B. Sequence-less input interaction with non-uniform values:

The following figure shows an example of sequence-less CIIT for non-uniform values:

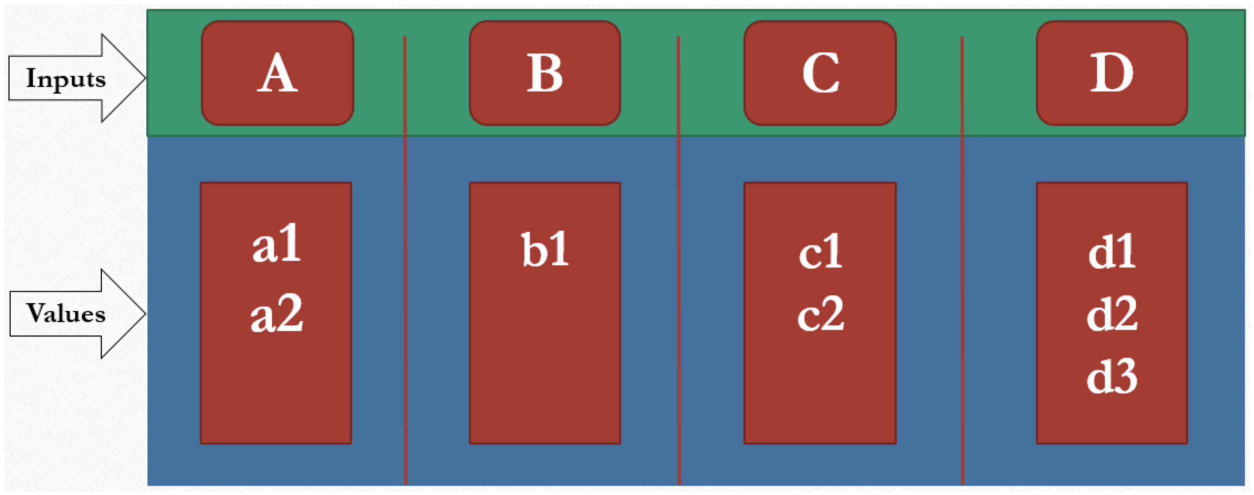


Fig 3.3: An example of sequence-less CIIT for non-uniform values.

The number of exhaustive test cases should be 2*1*2*3 = 12.

Now, the exhaustive test cases for this example are:

Table 3.3: Generated exhaustive test cases for **Fig 3.3**

No.	Exhaustive test case
1	[a1,b1,c1,d1]
2	[a1,b1,c1,d2]
3	[a1,b1,c1,d3]
4	[a1,b1,c2,d1]
5	[a1,b1,c2,d2]
6	[a1,b1,c2,d3]
7	[a2,b1,c1,d1]
8	[a2,b1,c1,d2]

9	[a2,b1,c1,d3]
10	[a2,b1,c2,d1]
11	[a2,b1,c2,d2]
12	[a2,b1,c2,d3]

In my research, I've worked on sequence-less CIIT for both uniform and non-uniform values using T-way tuple trees and this thesis focuses on making the tuple tree generation process much faster and more efficient to make the entire exhaustive test case minimization process faster.

3.2 A test scenario

The following figure is an example that shows how the discussed problem can be faced in real lfie:

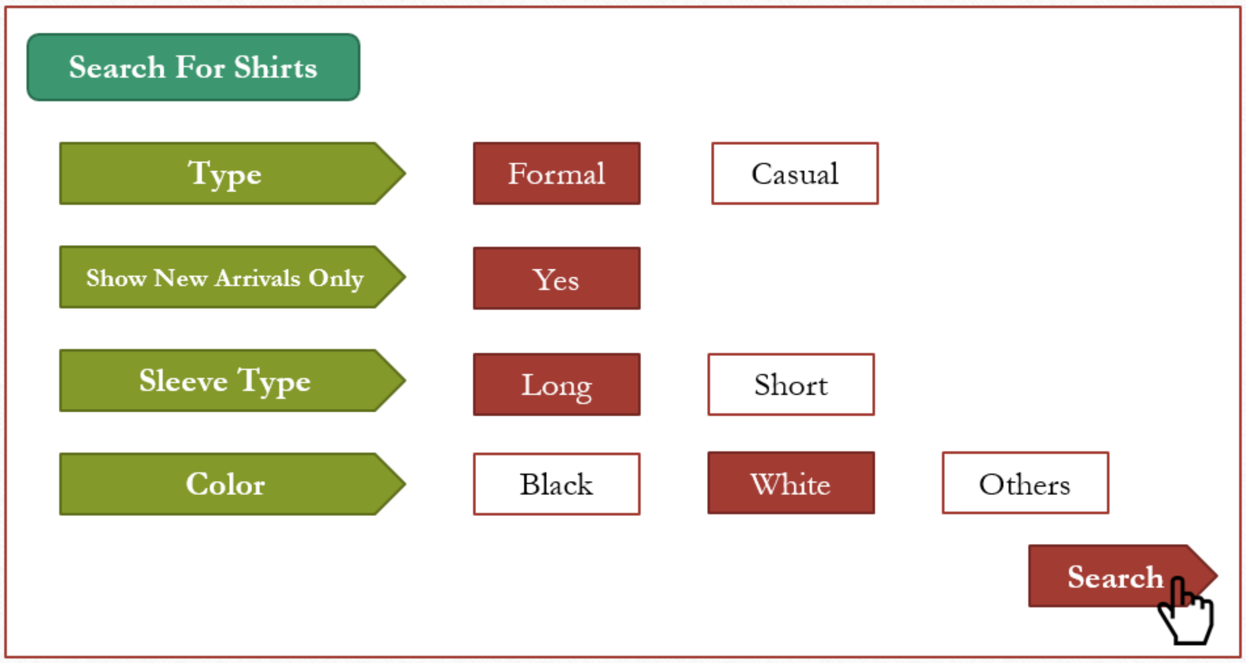


Fig 3.4: A test scenario

3.3 The test scenario formed as a problem

The test scenario from 3.2 can be described as the following figure below, where A = Type, B = Show New Arrivals Only, C = Sleeve Type, D = Color and a1 = Formal, a2 = Casual; b1 = Yes; c1 = Long, c2 = Short; d1 = Black, d2 = White, d3 = Others.

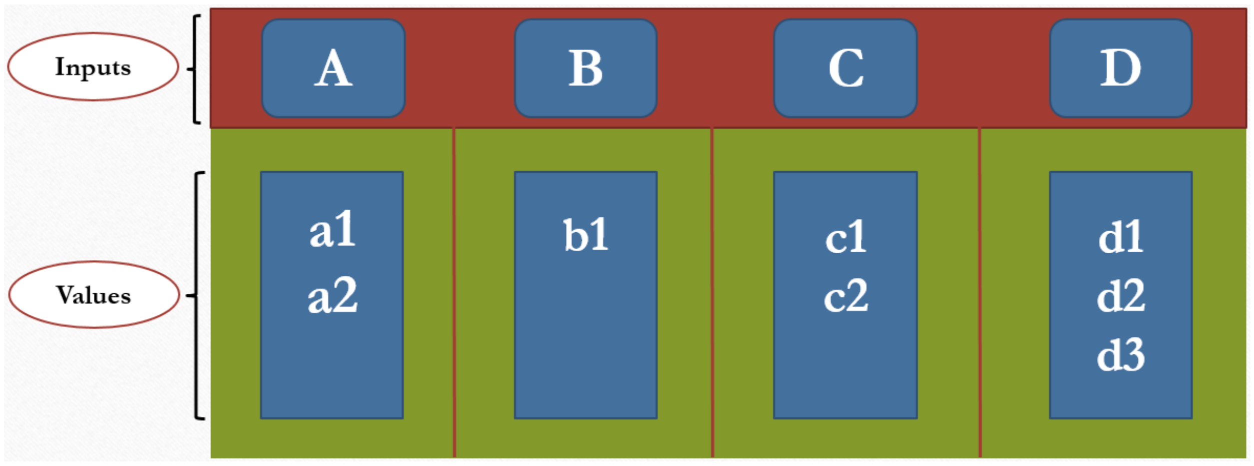


Fig 3.5: The test scenario formed as a problem

3.4 Exhaustive test cases for the scenario

The following figure shows the exhaustive test cases for the problem:

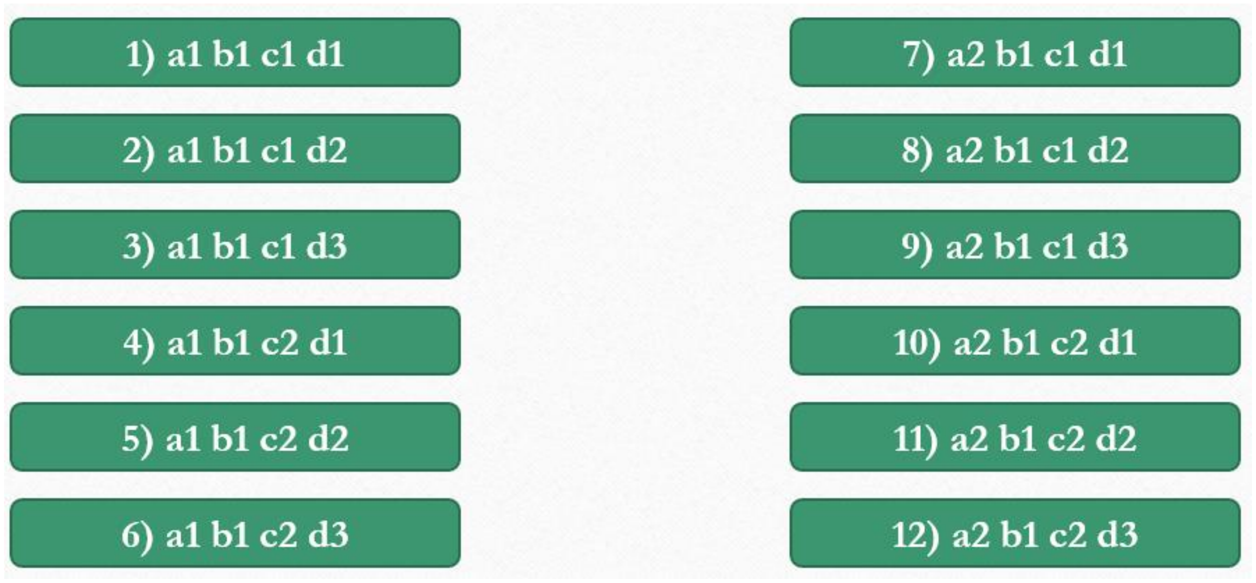


Fig 3.6: Exhaustive test cases

3.4 Representing the scenario as a tree

In the following figure, the entire scenario is represented as a tree. We have two options for initial node, either a1 or a2. The tree shows the entire path from the initial nodes to the leaf nodes.

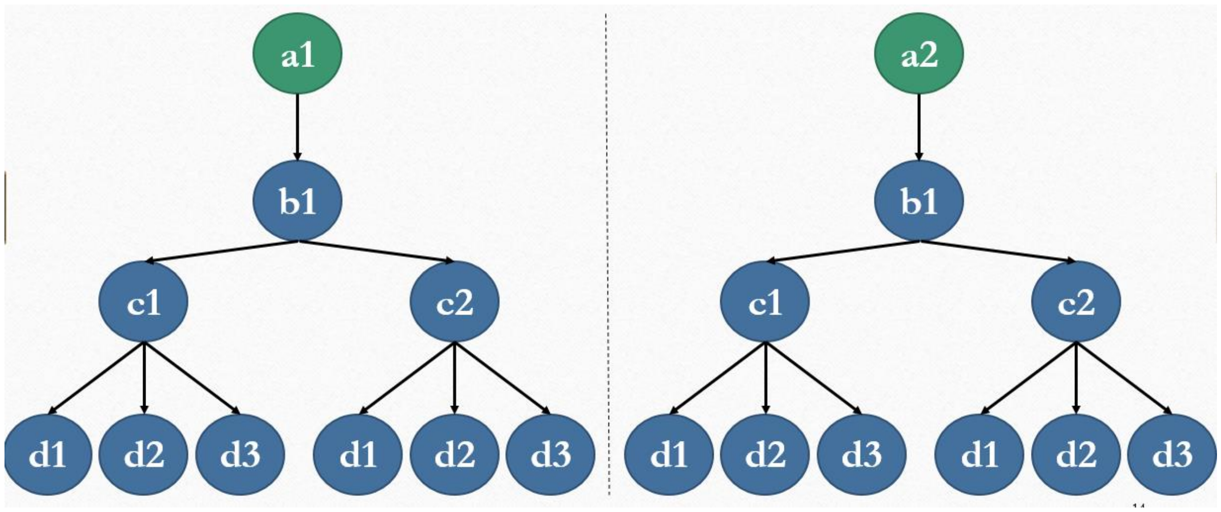


Fig 3.7: Tree representation of the scenario

3.5 Narrowing down the tree

The following figure is the narrowed down version of the tree in Fig 3.7:

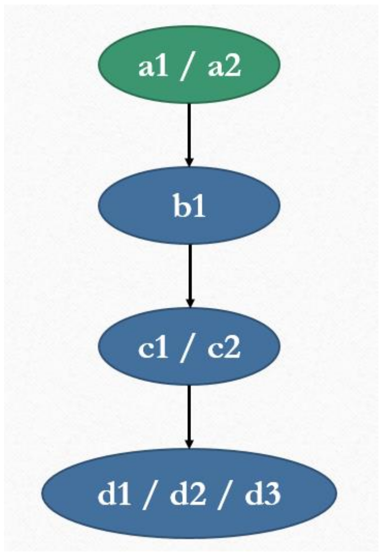
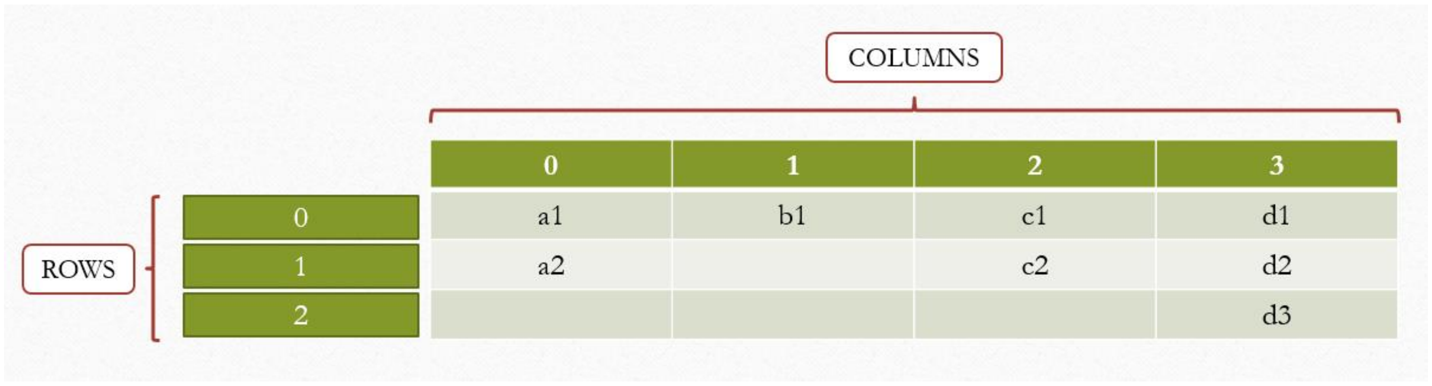


Fig 3.8: The narrowed down tree

3.6 Turning the scenario into a matrix

The technique that I used requires the problem to be represented into a matrix form first. The following figure shows how the problem is converted to a matrix:

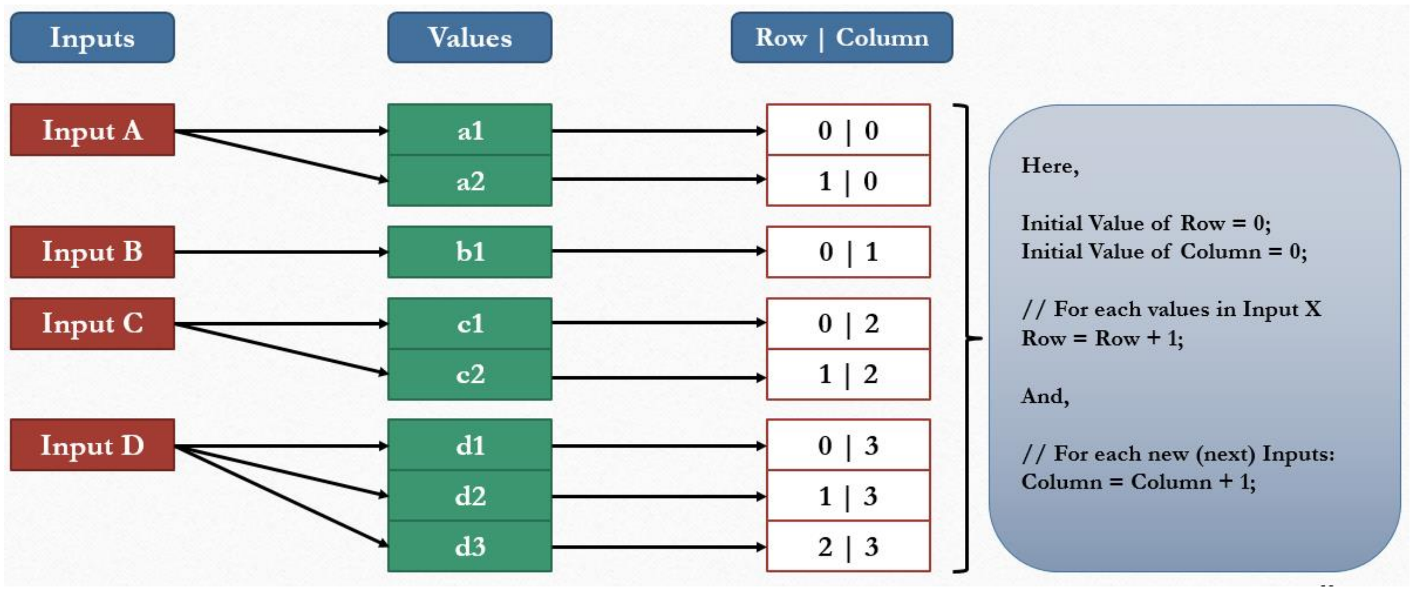


ROWS \ COLUMNS	0	1	2	3
0	a1	b1	c1	d1
1	a2		c2	d2
2				d3

Fig 3.9: Matrix representation of the scenario

3.7 Breaking down the matrix

The figures 3.10 and 3.11 below visualizes the explanation of the matrix:



Inputs	Values	Row | Column
Input A	a1	0 | 0
	a2	1 | 0
Input B	b1	0 | 1
Input C	c1	0 | 2
	c2	1 | 2
Input D	d1	0 | 3
	d2	1 | 3
	d3	2 | 3

```
Here,

Initial Value of Row = 0;
Initial Value of Column = 0;

// For each values in Input X
Row = Row + 1;

And,

// For each new (next) Inputs:
Column = Column + 1;
```

Fig 3.10: Break-down of the matrix from Fig 3.9

The following figure shows how the values are assigned:

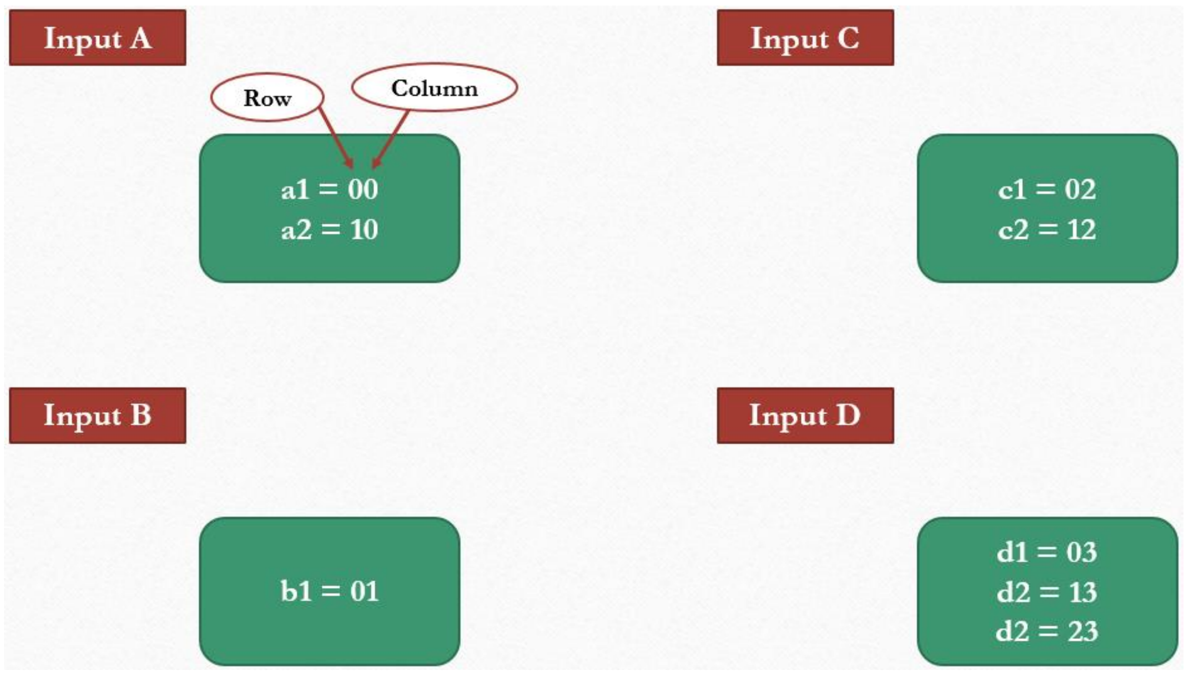
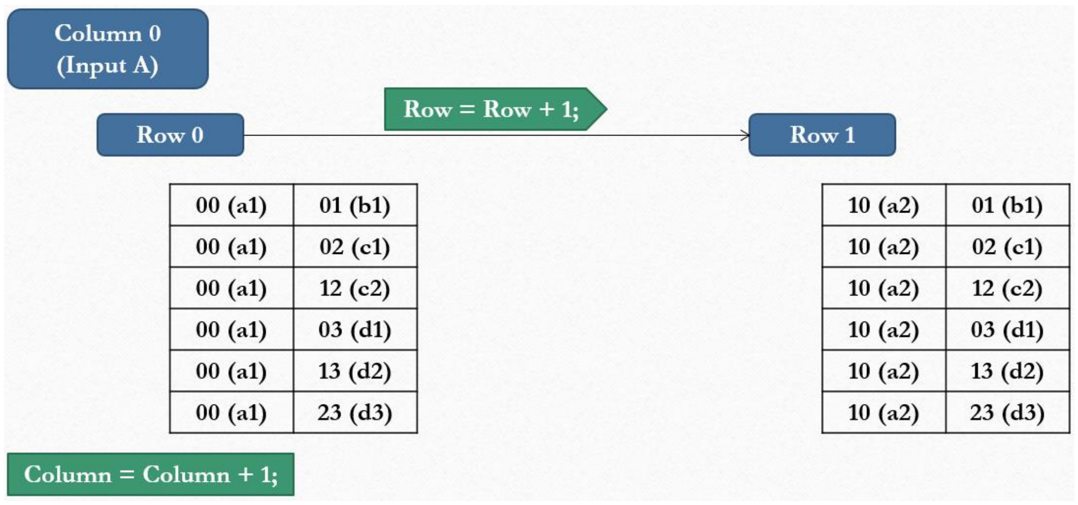


Fig 3.11: Assigning values from Fig 3.10

3.8 Tuples creation process (for T-ways, where T=2)

This section discusses how the tuples are created and the t-way tuple tree is formed.

3.8.1 Tuples creation process for Column 0 (Input A)



00 (a1)	01 (b1)
00 (a1)	02 (c1)
00 (a1)	12 (c2)
00 (a1)	03 (d1)
00 (a1)	13 (d2)
00 (a1)	23 (d3)

10 (a2)	01 (b1)
10 (a2)	02 (c1)
10 (a2)	12 (c2)
10 (a2)	03 (d1)
10 (a2)	13 (d2)
10 (a2)	23 (d3)

Fig 3.12: Tuple creation process for Column 0 (Input A)

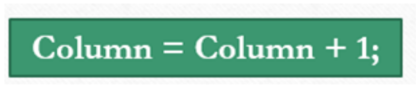
Explanation of Fig 3.12:

- Column 0 is the column for Input A.
- Input A has 2 values (a1, a2).
- So, a1 and a2 both are in the same column (Column 0).
- But a1 is in Row 0 and a2 is in Row 1.
- So, all the nodes that can be visited from a1, can also be visited from a2.

Thus, all the nodes can be visited from x can also be visited from y, z, … (where x, y, z, … all are in the same column).

3.8.2 Working with next columns

The next column is:



```
Column = Column + 1;
```

Therefore,

- We are in Column 1.
- Column 1 is the column for Input B.
- Input B has only 1 value (b1).

The following figure shows how the rest of the "to be visited" nodes are selected for rest of the columns starting from Column 1 (also includes the summary of the entire process):

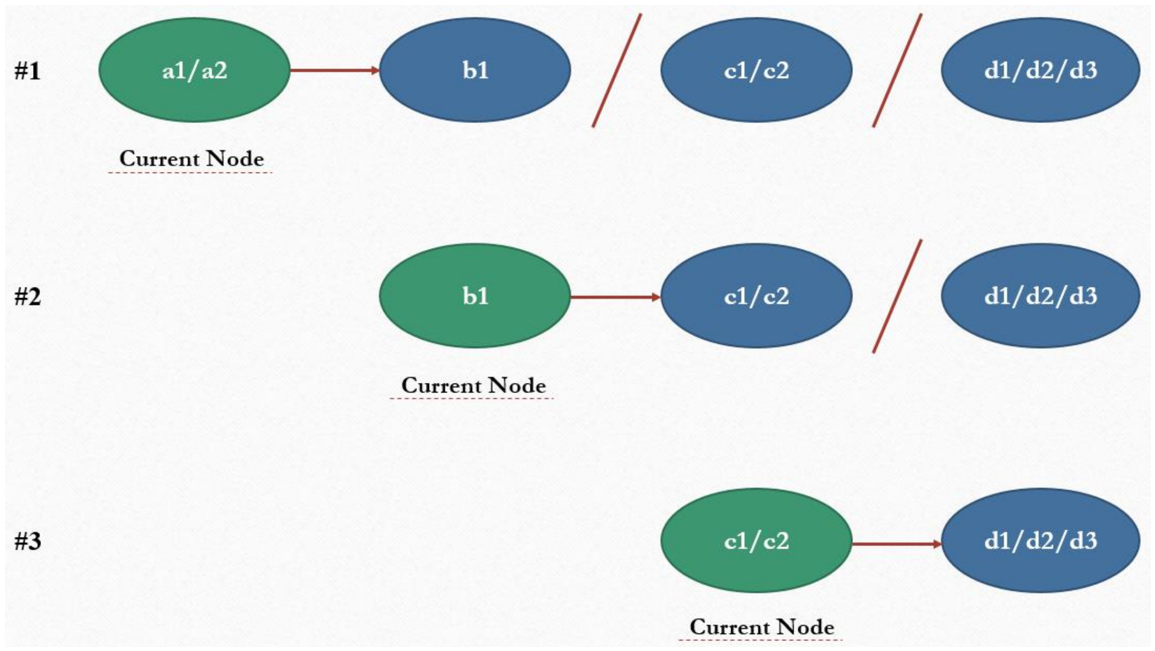


Fig 3.13: Summary of the entire tuple making process

Explanation of Fig 3.13:

1. For Column 0, the Current Node is either a1 or a2. As we have discussed earlier that, the "to be visited" nodes for the nodes in the same column, are same.
   So, if a1 or a2 is the Current Node, then the "to be visited" nodes are b1, c1, c2, d1, d2, d3.
   Process for Column 0 is finished. The next column is Column 1.

2. For Column 1, the Current Node is b1. If we pay attention we can see that, all the nodes that can be visited from the values of Column 0 (a1, a2), can be visited from the value from Column 1 (b1) except for the node in Column 1 itself, which is b1 itself in this case.
   So, if b1 is the Current Node, then the "to be visited" nodes are c1, c2, d1, d2, d3.
   Process for Column 1 is finished. The next column is Column 2.

3. For Column 2, is the same process as Column 1. Here, the Current Node is either c1 or c2.
   So, the "to be visited" nodes are d1, d2, d3.
   Process for Column 2 is finished. The next column is Column 3.

Column 3 will not be set as "Current" because:

- Total number of columns = 3
- The value of T in T-ways = 2
- So, the number of columns, from the end to towards the beginning, to not to set as "Current" is = (3 – 2) = 1
- The last one column is Column 3.
- Thus, Column 3 will not be set as "Current".

In conclusion,

This is the entire process of the algorithm that I have created and my research is based on. This algorithm makes the tuple tree generation process not only faster but also efficient. It also increases the number of inputs that can be worked with in a huge scale.

3.9 Enhanced T-way Tuple Tree Generator Algorithm (ETTTG Algorithm)

Algorithm 3.9: Enhanced T-way Tuple Tree Generator (ETTTG Algorithm) for sequence-less input interaction

Inputs: Position(pos), Counter(cnt)

Outputs: The “nodes to be visited” from the “current nodes”

Process:

```
1.  IF cnt = ways THEN
2.   Initialize paired values of Structure a to {0,0}
3.   FOR i = 0 to cnt - 1 DO
4.     Set numeric values of tuples into a
5.   END FOR
6.   Store values of a to Temp
7.   RETURN
8.  END IF
9.  IF pos > V THEN
10.  RETURN
11. END IF
12. FOR i = 1 to values[pos] DO
13.  Mark the current position as taken
14.  Increment pos by 1 and cnt by 1 and GOTO Step 1
15. END FOR
16. Increment pos by 1 and GOTO Step 1
```

End of Enhanced T-way Tuple Tree Generator

Fig 3.14: ETTTG Algorithm

3.10 Summary

The methodology, diagrams for visual representation and the algorithm is discussed here.

CHAPTER 4

EXPERIMENTAL RESULTS AND DISCUSSION

In Chapter 3, the methodology for this thesis is described and visualized theoretically. In this chapter, the algorithm's experimental results is discussed after implementing the algorithm and testing it by comparing it with other solution as well as by testing individually.

4.1 Input-output for the scenario of this thesis

Input Format:

```
--------------------------------------------------------------
Number of Inputs
Value of T (in T-way tuple tree)
Number of Values for Each Inputs
--------------------------------------------------------------
```

Example:

```
-------------------------------------------------------------
4
2
2 1 2 3
------------------------------------------------------------
```

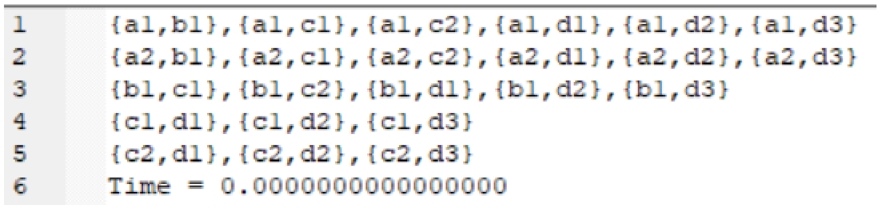
From given example,

Number of Inputs = 4 (means A, B, C, D)

Ways (T) = 2

Values of Inputs = A (a1, a2), B (b1), C (c1, c2), D (d1, d2, d3)

Output:



```
{a1,b1},{a1,c1},{a1,c2},{a1,d1},{a1,d2},{a1,d3}
{a2,b1},{a2,c1},{a2,c2},{a2,d1},{a2,d2},{a2,d3}
{b1,c1},{b1,c2},{b1,d1},{b1,d2},{b1,d3}
{c1,d1},{c1,d2},{c1,d3}
{c2,d1},{c2,d2},{c2,d3}
Time = 0.0000000000000000
```

Fig 4.1: Output for the scenario of the research

Discussion:

- Tuple trees with all the unique tuples are generated.
- Time taken = 0 ms (approx.)

4.2 Comparison with Generic Tuple Generator and Benchmarking

From observation, I've realized that none of the existing solutions focus on tuple tree creation faster and more efficient and focus on CA (Covering Array) and minimizing exhaustive test cases. The problems that deal with tuple creation process use generic way to create tuples and then apply algorithm for minimizing tuples on CA.

Thus, Generic Tuple Generator is denoted as GTG (ref: Othman, R. R. Design of a T-way Test Suite Generation Strategy Supporting Flexible Interactions. Universiti Sains Malaysia) and my proposed Enhanced T-way Tuple Tree Generator is denoted as ETTTG for the following comparison table.

Table 4.1: Comparison between GTG and ETTTG and Benchmarking

Input	Strategy	Time (approx.)	Success/Failure
(i) 6 2 2 1 2 3 1 2	ETTTG	0 ms	Success
	GTG	1 ms	Success
(ii) 8 2 2 1 2 3 1 2 4 1	ETTTG	0 ms	Success
	GTG	4 ms	Success
(iii) 10 3 2 1 2 3 1 2 4 1 3 1	ETTTG	0 ms	Success
	GTG	4 ms	Success

(iv) 10 2 2 1 2 3 1 2 4 1 3 1	ETTTG	0 ms	Success
	GTG	4 ms	Success
(v) 12 2 2 4 2 3 1 2 4 1 3 1 3 4	ETTTG	0 ms	Success
	GTG	n/a	Failure
(vi) 13 3 2 4 2 3 1 2 4 1 3 1 3 4 5	ETTTG	0 ms	Success
	GTG	n/a	Failure
(vii) 13 2 2 4 2 3 1 2 4 1 3 1 3 4 5	ETTTG	0 ms	Success
	GTG	n/a	Failure

4.3 Discussion of the result:

ETTTG obtains the following things:

- The fastest T-way strategy to generate tuple tree with unique tuples,
- The most efficient way to generate tuples, because:-
    i. When value of T increases, time complexity decreases,
    ii. The higher number of inputs make the better suit for this strategy compared to others, because ETTTG can handle maximum number of inputs among all other T-way tuple generation strategies.
    iii. It eliminates all kind of redundancy.

4.4 Summary

GTG can only handle few inputs to create unique tuples but proposed Enhanced T-way Tuple Tree Generator (ETTTG) can handle significant number of inputs and also the tuple creation process is much faster than any other existing tuple generation strategy. The main focus point of this thesis was to obtain a faster and more efficient T-way tuple tree generation process and to maximize the number of inputs to work with for input interaction. And from the results I can come to this conclusion that ETTTG achieved the goals of this thesis. In future, ETTTG can be used with other algorithms to minimize the number of exhaustive test cases in CIIT and also can make the process significantly faster and more efficient.

CHAPTER 5

CONCLUSION AND FUTURE WORKS

5.1 Findings and Contributions

5.1.1 Theoretical/Methodological Contribution

In this thesis, I’ve successfully created a new and enhanced algorithm for generating T-way tuple trees for T-way test case generation strategies. The algorithm is named Enhanced T-way Tuple Tree Generator (ETTTG) Algorithm.

ETTTG first turns the problem scenario into a matrix then solves the problem step by step and eliminates redundancy while generating the tuple tree which makes the tuple tree generation process faster and much more efficient.

5.1.2 Practical Implication

The significance of this algorithm and advantages over other algorithms is that, it works independently to generate T-way tuple trees of unique tuples and it is faster, efficient and can work with more inputs.

Thus, this algorithm can be used within a t-way test case generation strategy with an exhaustive test case minimization algorithm to get the best results.

5.2 Limitation

ETTTG can’t minimize the exhaustive test cases because it wasn’t designed to directly work on minimizing exhaustive test cases.

5.3 Recommendations for Future Works

This research purely focuses on the T-way tuple tree generation process and proposes an algorithm. This algorithm can be merged with another algorithm that uses T-way tuple trees to minimize number of exhaustive test cases in CIIT.

Therefore, the future work is to merge this algorithm with T-way test case generation strategy to minimize exhaustive test cases.

REFERENCES

Ahmed, B. S., Zamli, K. Z., & Lim, C. P. (2012a).

Application of particle swarm optimization to uniform and variable strength covering array construction. Applied soft computing, 12(4), 1330-1347. doi: 10.1016/j.asoc.2011.11.029.

Afzal, W., Torkar, R. & Feldt, R. (2009).

A systematic review of search-based testing for non-functional system properties. Information and software technology, 51(6), 957-976.

Ahmed, B. S., Zamli, K. Z. and Lim, C. P. (2012b).

Constructing a t-way interaction test suite using the particle swarm optimization approach. International journal of innovative computing, information and control (ICIC), 8(1(A)), 431-451.

Ahmed, B. S., & Zamli, K. Z. (2010).

T-way test data generation strategy based on particle swarm optimization. In proceedings of 2nd international conference on computer research and development, 93-97. doi: 10.1109/iccrd.2010.56.

Alsewari, A. A., & Zamli, K. Z. (2011).

Interaction test data generation using harmony search algorithm. In proceedings of the IEEE symposium on industrial electronics and applications, 559-564. doi: 10.1109/isiea.2011.6108775

Alsariera, Y. A., & Zamli, K. Z. (2015).

A bat-inspired strategy for t-way interaction testing. Journal of advanced science letters, 21(8), 2281-2284. doi: 10.1166/asl.2015.6316.

Arshem, J. (2004). Test vector generator.
Retrieved on April 5, 2017 from http://sourceforge.net/projects/tvg.

Alsewari, A. A., & Zamli, K. Z. (2014).

An orchestrated survey on t-way test case generation strategies based on optimization algorithms. In proceedings of the 8th international conference on robotic, vision, signal processing & power applications, 255-263. doi: 10.1007/978-981-4585-42-2_30.

Bryce, R. C., & Colbourn, C. J. (2009).

A density-based greedy algorithm for higher strength covering arrays. Software testing, verification and reliability, 19(1), 3753. doi:10.1002/stvr.393.

Cohen, D., Dalal, S., Fredman, M., & Patton, g. (1997).

The AETG system: an approach to testing based on combinatorial design. IEEE transactions on software engineering, 23(7), 437-444. doi:10.1109/32.605761

Cohen, D., Dalal, S., Kajla, A., & Patton, G. (1994).

The Automatic Efficient Test Generator (AETG) system. In proceedings of the IEEE international symposium on software reliability engineering, 303-309. doi:10.1109/issre.1994.341392

Cohen, M. B. (2004).

Designing test suites for software interaction testing. University of Auckland.

Cohen, M. B., Colbourn, C. J., & Ling, A. C. (2003a).

Augmenting simulated annealing to build interaction test suites. In proceedings of the 14th international symposium on software reliability engineering (ISSRE'03), 394-405. doi:10.1109/issre.2003.1251061

Cohen, M., Gibbons, P., Mugridge, W., & Colbourn, C. (2003b).

Constructing test suites for interaction testing. In proceedings of the 25th international conference on software engineering, 38-48. doi:10.1109/icse.2003.1201186.

Cohen, M. B., Colbourn, C.J. & Ling, A. C. H. (2008).

Constructing strength three covering arrays with augmented annealing. Discrete Math, 308, 2709–2722.

Črepinšek, M., Liu, S., Mernik, L., & Mernik, M. (2014a).

Is a comparison of results meaningful from the inexact replications of computational experiments? Soft computing, 20(1), 223-235. doi: 10.1007/s00500-014-1493-4.

Črepinšek, M., Liu, S-H., & Mernik, L. (2014b).

Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them. Applied soft computing, 19, 161-170. doi:10.1016/j.asoc.2014.02.009.

D. Yazdani, S. Sadeghi-Ivrigh, D. Yazdani, A. Sepas-Moghaddam and M. R. Meybodi.

Fish Swarm Search Algorithm: A New Algorithm for Global Optimization, International Journal of Artificial Intelligence, vol. 13, no. 2, pp. 17-45, 2015

Draa, A. (2015).

On the performances of the flower pollination algorithm - qualitative and quantitative analyses. Applied soft computing, 34, 349-371. doi:10.1016/j.asoc.2015.05.015.

Hartman, A., Klinger T., Raskin L. (2005).

IBM intelligent test case handler. Retrieved on April 5, 2017 from http://ibm-intelligent-test-case-handler.updatestar.com/en.

Harrold, M. J. (2000).

Testing: A roadmap. In proceedings of the conference on the future of software engineering, 61-72.

Jenkins, B. (2005). Jenny test tool.

Retrieved on April 5, 2017 from http://www.burtleburtle.net/bob/math/jenny.html.

Kennedy, J. & Eberhart, R. (1995a).

Particle swarm optimization. In proceedings of the IEEE international conference on neural networks, 1942-1948. doi: 10.1109/ICNN.1995.488968.

Kennedy, J. and Eberhart, R. (1995b).

A new optimizer using particle swarm theory. In proceedings of the 6th international symposium on micro machine and human science, 39-43. dio: 10.1109/MHS.1995.494215.

Kuhn, D. R., Kacker, R. N., Lei, Y. (2010).

Practical combinational testing. U.S. department of commerce, national institute of standards and technology (NIST), Special publication 800-142.

Lei, Y., Kacker, R., & Kuhn, D. R. (2007a).

IPOG: A general strategy for t-way software testing. In proceedings of the 14th annual IEEE international conference and workshops on the engineering of computer-based systems (ECBS'07), 549-556. doi: 10.1109/ECBS.2007.47.

Lei, Y., Kacker, R., Kuhn, R., Okun, V., & Lawrence, J. (2007b).

IPOG/IPOGD: Efficient test generation for multi-way combinatorial testing. Journal of software testing, verification and reliability, 18(3), 125-148.

Mernik, M., Liu, S., Karaboga, D., & Črepinšek, M. (2015).

On clarifying misconceptions when comparing variants of the artificial bee colony algorithm by offering a new implementation. Information Sciences, 291, 115-127. doi: 10.1016/j.ins.2014.08.040.

Mahmud, T. & Ahmed, B. S. (2015).

An effective strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software functional testing Use. Expert system with application, 42, 8753-8765.

Morgan, P., Hambling, B., Thompson, G., Samaroo, A., Williams, P. (2015).

Software testing: an istqb-bcs certified tester foundation guide. BCS learning & development limited, USA. ISBN 1780172990, 9781780172996.

Nasser, A. B., Alsewari, A. A., & Zamli, K. Z. (2015).

Tuning of Cuckoo Search based Strategy for T-way Testing. ARPN Journal of Engineering and Applied Sciences, 10(19), 8948-8953.

Nuraminah R., R. R. Othman*, Zahereel I.A.K., Muzammil J., (2017).
A Review on Recent T-way Combinatorial Testing Strategy, DOI: 10.1051/matecconf/201714001016

Nie, C., Xu, B., Shi, L., & Dong, G. (2005).

Automatic Test Generation for N-Way Combinatorial Testing. Lecture Notes in Computer Science Quality of Software Architectures and Software Quality, 203-211. doi: 10.1007/11558569_15.

Othman, R. R. (2012).

Design of a T-way Test Suite Generation Strategy Supporting Flexible Interactions. Universiti Sains Malaysia (USM).

Othman, R. R. and Zamli, K.Z., 2011.

T-way strategies and its applications for combinatorial testing. International journal on new computer architectures and their applications (IJNCAA), 1(2), 459-473.

Shiba, T., Tsuchiya, T. and Kikuno T. (2004).

Using artificial life techniques to generate test cases for combinatorial testing. In proceedings of the 28th annual international computer software and applications Conference, 01, 72–77.

Tassey, G. (2002).

The economic impacts of inadequate infrastructure for software testing. National institute of standards and technology, RTI Project Number 7007.011.

Williams, A. W. (2000).

Determination of test configurations for pair-wise interaction coverage. In proceedings of the advances in information and communication technology testing of communicating systems, 59-74. doi: 10.1007/978-0-38735516-0_4.

Williams, A. W. & Probert, R. L. (2001).

A measure for component interaction test coverage. In proceedings of the International conference on computer systems and applications (AICCSA 2001), 304-311.

Younis, M. I., & Zamli, K. Z. (2011).

MIPOG-an efficient t-way minimization strategy for combinatorial testing. International journal of computer theory and engineering, 3(3), 388-397.

Zamli, K. Z., Din, F., Kendall, G., & Ahmed, B. S. (2017).

An experimental study of hyper-heuristic selection and acceptance mechanism for combinatorial t -way test suite generation. Information sciences, 399, 121-153. doi: 10.1016/j.ins.2017.03.007.

Zamli, K. Z., Alkazemi, B. Y., & Kendall, G. (2016).

A Tabu Search hyper-heuristic strategy for t-way test suite generation. Applied soft computing, 44, 57-74. doi:10.1016/j.asoc.2016.03.021.

Zamli, K. Z., Klaib, M. F., Younis, M. I., Isa, N. A., & Abdullah, R. (2011).

Design and implementation of a t-way test data generation strategy with automated execution tool support. Information Sciences, 181(9), 1741-1758. doi:10.1016/j.ins.2011.01.002.