A BAT-INSPIRED T-WAY STRATEGY FOR MIXED-STRENGTH TEST SUITE GENERATION

YAZAN AHMAD AL SARIERA

Doctor of Philosophy

UNIVERSITI MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name : Yazan Ahmad Sadeq Al Sariera

Date of Birth : 24/09/1988

Title : A Bat-inspired T-Way Strategy for Mixed-Strength Test Suite Generation

Academic Session : 2017/2018

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)	(Supervisor's Signature)
M858546	Prof. Dr. Kamal Z. Zamli
New IC/Passport Number	Name of Supervisor
Date: 27 October 2017	Date: 27 October 2017

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

SUPERVISOR'S DECLARATION

We hereby declare that we have checked this thesis and in our opinion, this thesis is adequate in terms of scope and quality for the award of the degree of Doctor of Philosophy.

_____

(Supervisor's Signature)

Full Name : Prof. Dr. Kamal Z. Zamli

Position : Dean, Faculty of Computer Systems and Software Engineering, (Professor)

Date : 27 October 2017

_____

(Co-supervisor's Signature)

Full Name : Assoc. Prof. Mazlina Abdul Majid

Position : Deputy Dean, Faculty of Computer Systems and Software Engineering, (Associate Professor)

Date : 27 October 2017

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 : Yazan Ahmad Sadeq Al Sariera

ID Number : PCS14001

Date : 27 October 2017

A BAT-INSPIRED T-WAY STRATEGY FOR MIXED-STRENGTH TEST SUITE GENERATION

YAZAN AHMAD SADEQ AL SARIERA

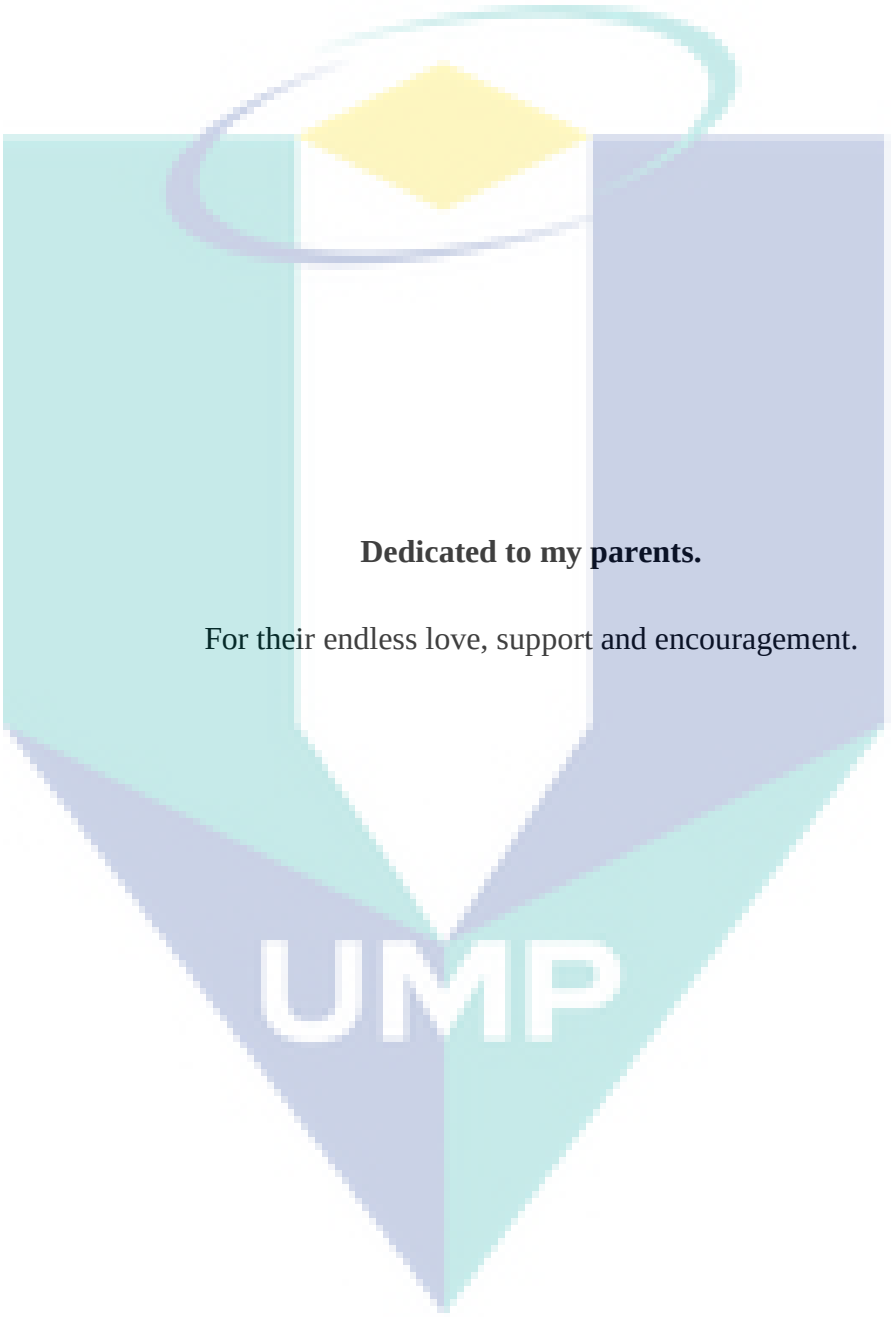
Thesis submitted in fulfillment of the requirements
for the award of the degree of
Doctor of Philosophy

Faculty of Computer Systems and Software Engineering

UNIVERSITI MALAYSIA PAHANG

OCTOBER 2017

DEDICATION



Dedicated to my parents.

For their endless love, support and encouragement.

ACKNOWLEDGEMENTS

First and foremost, I owe my deepest gratitude to Allah (SWT) for giving me live, strength and guidance throughout my study period and the ability to complete this thesis.

I am deeply grateful to my parents: my mother Eman Alsariera and my father Ahmad Alsariera for your prayers and unlimited support. I certainly would not be where I am today without their nurture, guidance, love and care throughout my life. There are not enough words in the world to express my appreciation. Whatever I do, I will not be able to return the love, bestowal and support. I pray Allah for your perfect health and prolong age and may Allah reward you richly. Love you both.

I am greatly indebted and appreciate very much to my brothers and sister for their encouragements, supports and sacrifices all the time. To my relatives and dearest beloved, a grateful thank you for your support and encouragement, may Allah bless you all.

I would like to express my gratitude to my research supervisor Prof. Dr. Kamal Zuhairi Zamli for his excellent ideas, profound technical vision, invaluable guidance, patience, motivation, many fruitful discussions and constant support in making this thesis possible. I am so much grateful for accepting me to become a part of his research team.

I would like to thank my co-supervisor Assoc. Prof. Mazlina A. Majid and Dr. AbdulRahman A. AlSewari for their support and the faculty members and staff for their hard work as well.

Finally, I would like to acknowledge and thank Universiti Malaysia Pahang for allowing me to conduct my research and providing the required assistance. In addition, I would like to thank the Ministry of Science, Technology and Innovation (MOSTI), Malaysia for funding this work.

ABSTRAK

Pengujian perisian merupakan aktiviti penting dalam kitar hayat pembangunan perisian Walaubagaimanapun, ujian menyeluruh untuk perisian berkonfigurasi tinggi adalah tidak praktikal kerana kekangan masa dan sumber. Tambahan pula, ujian menyeluruh membawa kepada masalah letupan kombinasi di mana kes-kes ujian berkembang dengan pesat dengan peningkatan input perisian. Kerana keberkesanannya bagi mengesan pepijat, ramai penyelidik kini beralih kepada strategi persampelan berdasarkan interaksi input, yang dipanggil ujian *t*-hala, di mana *t* menunjukkan kekuatan interaksi. Dikenali sebagai masalah NP-lengkap (iaitu, tidak berketentuan Polinomial masa), proses mengurangkan kes-kes ujian *t*-hala amat mencabar kerana ruang carian yang luas apabila berurusan dengan nilai-nilai input besar. Setakat ini, banyak strategi *t*-hala yang telah dicadangkan dalam literatur. Baru-baru ini, para penyelidik telah mencadangkan penggunaan meta-heuristik berasaskan strategi *t*-hala seiring dengan kemunculan bidang baru dipanggil Kejuruteraan Perisian berasaskan Pencarian (SBSE). Walaupun berguna, tidak ada strategi *t*-hala berdasarkan meta-heuristik tunggal boleh mengatasi yang lain. Atas sebab ini, pencarian strategi *t*-hala meta-heuristik baru masih didambakan. Tesis ini membentangkan reka bentuk dan pelaksanaan yang strategi t-hala meta-heuristik baru, yang dikenali strategi *t*-hala kelawar (BTS), untuk menjana kekuatan campuran *t*-hala sut ujian. BTS adalah strategi *t*-hala pertama yang menggunakan algoritma kelawar sebagai teras dan mengadaptasi jarak Hamming sebagai kriteria pemilihan akhir bagi menambahbaik explorasi penyelesaian baharu. Keputusan eksperimen disokong oleh analisis statistik bukan parametrik menunjukkan bahawa BTS memberikan prestasi daya saing yang kompetitif berbanding strategi-strategi lain. Khususnya, BTS telah mencapai dan memadankan 68.181% saiz terbaik dari eksperiment penanda aras disamping menghasilkan 32.575% saiz terbaik baru. Penemuan ini menyumbang kepada bidang pengujian perisian dengan mengurangkan bilangan kes pengujian perisian untuk larian.

ABSTRACT

Software testing is essential part of software development life cycle. Yet, exhaustive testing of highly configurable software is impractical owing to the limited time and resources. Furthermore, exhaustive testing leads to a combinatorial explosion problem whereby the test cases grow exponentially with the increase of software inputs. Owing to its effectiveness for bug finding, many researchers are turning to the sampling strategies based on input interaction, called *t*-way testing, where *t* indicates the interaction strength. Known to be an NP-complete (i.e. Non-deterministic Polynomial-time) problem, the process of minimizing *t*-way test cases is challenging owing to the potentially large generated search space when dealing with large input values. To date, many *t*-way strategies have been proposed in the literature. Recently, researchers have advocated the adoption of meta-heuristic based *t*-way strategies in line with the emergence of the new field called Search Based Software Engineering (SBSE). Although helpful, no single meta-heuristic based *t*-way strategies can claim dominance over their other counterparts. For this reason, the search for a new meta-heuristic based *t*-way strategy is still a useful endeavor. This thesis presents the design and implementation of a new meta-heuristic based t-way strategy, called Bat-inspired *t*-way Strategy (BTS), for generating a mixed-strength t-way test suite. BTS is the first *t*-way strategy that adopts the Bat-inspired algorithm as its core implementation and adopts the Hamming distance as the final selection criteria to enhance the exploration of new solution. The experimental results supported by non-parametric statistical analysis demonstrate that BTS gives competitive performance over its counterparts. Specifically, BTS has achieved and matched 68.181% of the best sizes from the published benchmark results with 32.575 % new known best sizes. This finding contributes to the field of software testing by minimizing the number of test cases for test execution.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

A_0	Loudness
A_i	Initial loudness
ABC	Artificial Bee Colony
ABC- CAG	Artificial Bee Colony-Covering Array Generator
ACA	Ant Colony Algorithm
ACA-Shiba	Ant Colony Algorithm implemented by Shiba
ACO	Ant Colony Optimization
ACS	Ant Colony System
ACS-Chen	Ant Colony Algorithm implemented by Bryce
ACS-VSITs	Ant Colony System Variable Strength Interaction Test suites
ACTS	Advanced Combinatorial Testing Suite
AETG	Automatic Efficient Test Generator
ANNs	Artificial Neural Networks
AR	Anti-random
BA	Bat-inspired Algorithm
BA*	Bees Algorithm
BC	Base Choice
BCAETG	Compound BC and AETG strategy
BE	Binary Element
BES	Binary Element Set
BKM	Bat-K-Means
BTS	Bat-inspired Testing Strategy
CA	Covering Array
CASA	Simulated Annealing Algorithm for constrained Combinatorial interaction testing
CATS	TestCover
CEG	Cause and effect graphing
CI	Confidence interval
CPU	Central Processing Unit
CS	Cuckoo Search
CTE-XL	Classification-Tree Editor eXtended Logics
CTM	Classification-Tree Method
CTS	Combination t-Tuple Set
CTS	Combinations t-Tuples Set
CTS*	Combinatorial Test Services
CNC	Computer Numerical control
DF	Degree Of Freedom
DPSO	Discrete Particle Swarm Optimization
E	Element
EC	Element Combination
EC's	Element Combinations
EGA	Evolutionary Genetic Algorithm
ES	Element Set
FPA	Flower Pollination Algorithm

FSAPSO	Fuzzy Logic Agorithm Particle Swarm Optimization
FTS	Final Test Suite
GA	Genetic Algorithm
GA-Huang	Genetic Algorithm strategy implemented by Huang
GAPTS	Genetic Algorithm for Pairwise Test Sets
GUI	Graphical User Interface
H_0	Null hypothesis
H_1	Alternative hypothesis
HC	Hill Climbing
HC-Bryce	Hill Climbing strategy implemented by Bryce
HHH	High Level Hyper-Heuristic
HS	Harmony Search
HS-PTSGT	Harmony Search-Pairwise Test Suite Generator Tool
HSS	Harmony Search Strategy
HSTSG	Harmony Search Test Suite Generator
IE	Interaction Element
IE's	Interaction Elements
IET	Interaction Element Tuples
HIS	Improved HS
IPO	In Parameter Order
IPOG	In-Parameter-Order-General
IPOG-D	In-Parameter-Order-General Double
ISA	Improved SA
ITCH	Intelligent Test Case Handler
JRE	Java Running Environments
LAHC	Late Acceptance Hill Climbing
LOC	Lines of Code
mAETG	Modified Automatic Efficient Test Generator
MCA	Mixed-level Covering Array
mMCA	Mixed-strength Mixed-level Covering Array
MIPOG	Modified IPOG
MOCell	A Cellular genetic algorithm for MultiObjective optimization
MOL	Many Optimization Liaisons
mCA	Mixed-strength Covering Array
mTCG	Modified Test Case Generator
NP	Non-deterministic Polynomial-time
NSGA-II	Nondominated Sorting Genetic Algorithm II
OA	Orthogonal Arrays
OATS	Orthogonal Array Based Testing Strategy
P	Parameter
PGAS	A Parallel Genetic Algorithm based on Spark
PHSS	Pairwise Harmony Search Strategy
PICT	Pairwise Independent Combinatorial Testing
PITS	Prioritized pairwise Interaction Test Suite
PPSTG	Pairwise Particle Swarm Test Generator
PPW	Partly Pair-Wise

PSO	Particle Swarm Optimization
PSO-Chen	Particle Swarm Optimization strategy implemented by Chen
PSTG	Particle Swarm Test Generator
PTSG-GA	Pairwise test set generator using genetic algorithm
Q	Frequency
r	Emission of pulse rate
RAM	Random Access Memory
RTS	Reverse Tracking Strategy
SA	Simulated Annealing
SA-Bryce	Simulated Annealing strategy implemented by Bryce
SA-Mayer	Simulated Annealing strategy implemented by Mayer
SAVNS	Simulated Annealing Variable Neighbourhood Search
SBC	Simulated Bee Colony
SBSE	Search-Based Software Engineering
SSO	Simplified Swarm Optimization
T	Interaction strength
tCA	Improved CASA
TCG	Test Case Generator
TConfig	Test Configuration
T_{max}	Number of generation (iteration)
TS	Tabu search
ts	Time Step
TS-Bryce	Tabu Search strategy implemented by Bryce
TVG	Test Vector Generator
v	Value
MC	Mixed-strength condition
VSITs	Variable Strength Interaction Test suites
VS-PSTG	Variable Strength Particle Swarm Test Generator
X	Position (location)
X^2	Chi-square
a	Alpha

CHAPTER 1

INTRODUCTION

Software and hardware are the main components that drive computer technologies. Unlike hardware, software does not wear out. Here, software is a set of written code, functions and procedures that enables the user to accomplish a specific task. Whenever possible, software can be the replacement for its hardware counterparts, because software is flexible and allows easy customization as needed. In addition, the use of software can help to control maintenance costs.

Software development passes through several stages, called the software development life cycle. Generally, the activities in the software development life cycle are divided into two main processes: building the product (*creating the software*) and maintaining the product quality (Baresi & Pezzè, 2006). Every single cycle in software development must meet the highest production standard to ensure software quality, in order to cope with software faults and defects (Naik & Tripathy, 2008).

Software testing is the main gatekeeper of software quality, that is, in terms of minimizing the risk of software failure. Specifically, software testing ensures that software meets its specifications and quality standards.

1.1 Overview of Software Testing

Software testing is an integral part of the software development life cycle that consumes more than 40 to 50% of the development costs (Bertolino, 2007; Carroll, 2003; Kaner et al., 1999; Pan, 1999; Pendharkar, 2010). Often represented as a single activate in the development life cycle, software testing consists of a series of planned tasks that need to be executed along with the software development activities to ensure that a product is delivered without any defects (Katherine & Alagarsamy, 2012). Figure 1.1 shows the overall picture of a general software testing life cycle.

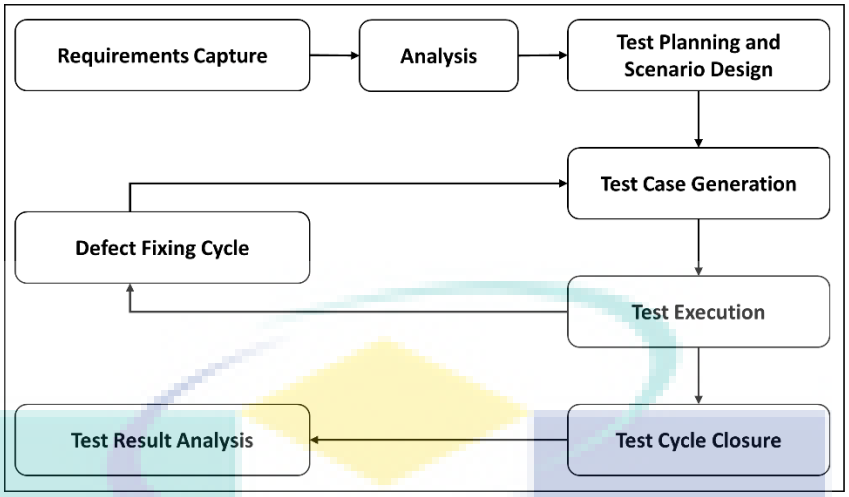


Figure 1.1 Overview of the software testing life cycle.
Source: Katherine (2012).

Referring to Figure 1.1, the software testing life cycle starts with the requirements capturing task. Here, the test engineers interact with the software-under-test specifications to capture the procedures and requirements of the software. Based on the captured requirements, software and test engineers collaborate to design test scenarios to prepare test cases that cover the entire input parameters of the software-under-test. Then, test engineers execute the generated test cases against the software-under-test. In case of defects detection, test engineers and developers collaborate to fix the detected defects with the support of software engineers. After that, the test execution is completed. The requirements' engineers confirm the results to ensure that the software-under-test is meeting its specification.

Concerning the test generation stage, manual test cases generation becomes practically impossible. Likewise, automated testing for all the possible inputs of software configurations (known as *exhaustive testing*) is impracticable due to the time and resource constraints. Therefore, effective sampling strategy can be an alternative to exhaustive testing which can reveal defects in the software-under-test (Burnstein, 2006; Hass, 2008).

Over time, test case sampling (or *design*) strategies (i.e. cause and effect graphing, equivalence class partitioning, boundary value analysis, systematic random testing) have emerged to generate a set of test cases that are capable of detecting software faults and defects of the software-under-test. Although useful, the abovementioned strategies do not sufficiently address the interaction between software inputs. For this reason, researchers

have proposed a new sampling approach based on interaction testing, termed *t*-way testing (where *t* indicates the interaction strength) (Kuhn et al., 2004; Kuhn et al., 2008). Specifically, *t*-way testing focuses on the faults that occur due to the interaction between two or more parameters of the software of interest.

1.2 Research Motivation

Software systems are becoming more complex owing to the improvement of computer power as well as sophisticated and complex demands on technology. These complexities are sufficiently intricate and can often cause unwanted quality and reliability issues amongst the software components.

Although desirable, exhaustive testing is impractical as the number of test cases can be tremendously large (Chaudhuri & Zhu, 1992; Copeland, 2004; Roper, 2002) even for the simplest software systems. As illustration, consider a simple logo generating system for a Computer Numerical Control (CNC) machine that support both laser printing and laser engraving with multi-color profile. The CNC software system can be seen in Figure 1.2.

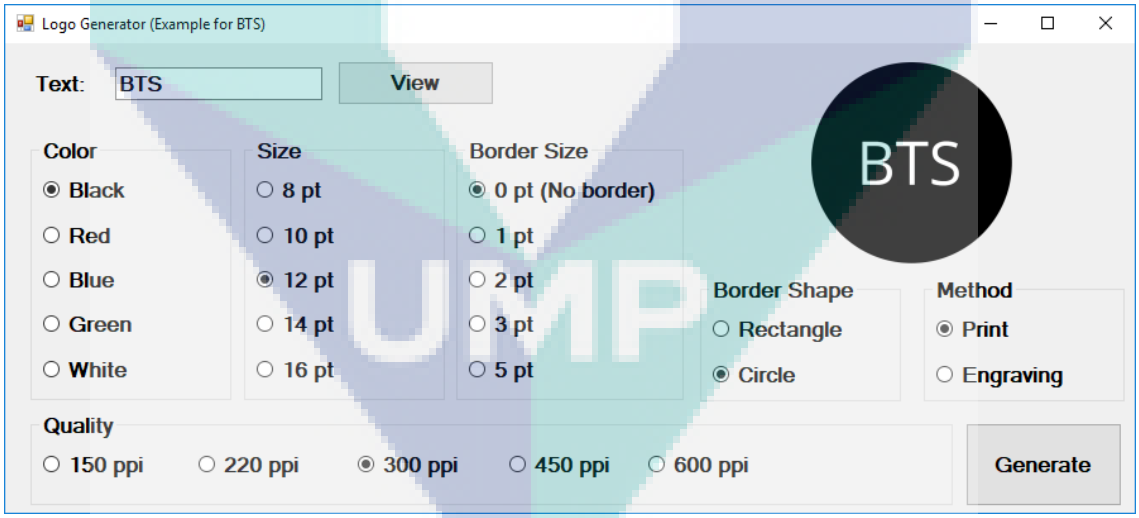


Figure 1.2 The CNC software system.

The options dialog consists of one text box that accepts two values either "with text" or an empty, six radio selectors groups - four with five values each and another two radio selectors with two values each. Therefore, the system consists of four parameters of five-values each and three parameters of two-values, Here, each group of parameters have similar number of dependencies. Thus, testing all the inputs configurations (i.e.,

exhaustive test) would require $5^4 \times 2^3 = 5{,}000$ test cases. Assuming that the four of the parameters (i.e. text, border shape, method and quality) have more impact on the overall system, the exhaustive test for the four aforementioned parameters would require $2^3 \times 5^1 = 40$ test cases. Thus, the system overall exhaustive test would involve at most $5000 + 40 = 5040$ test cases. If each test case required 5 minutes, then the testing process would approximately require 17.5 days to complete the exhaustive testing process of all the possible configurations. This example emphasizes that testing all the possible software inputs exhaustively is impossible owing to the limited time and resources for testing. Utilizing mixed-strength *t*-way test generation as illustrated in Figure 1.3, the overall system can be tested for $t = 2$ and the four mixed-strength parameters for $t = 3$. In this case, the overall test suite can be minimized up to 50 test cases. More explanation on *t*-way and mixed-strength test suite generation will be provided in Section 2.3 in Chapter 2.

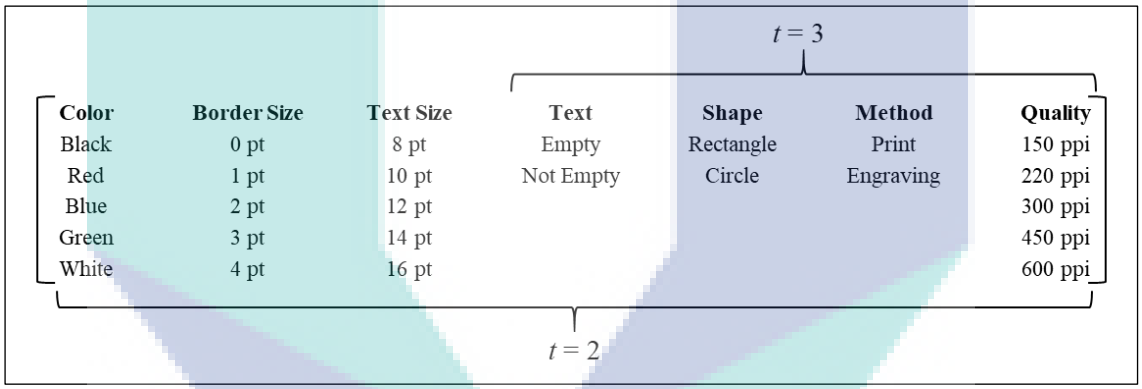


Figure 1.3 The parameters and their values for CNC software system.

The aforementioned example has illustrated the potential test case minimization for a simple software with small parameters and values. Considering complex and highly configurable software system, the number of test cases for testing consideration that can be larger owing to the presence of many parameters and values (Chaudhuri & Zhu, 1992; Copeland, 2004; Roper, 2002).

1.3 Problem Statements

Software systems are getting more complex owing to the advancement of computer technologies as well as sophisticated and complex demands from the users. These complexities are sufficiently intricate and can often cause unwanted quality and reliability issues amongst the systems components. In this regard, software testing can be considered to be an essential part of the development process (Bryce et al., 2005) to ensure that all software requirements specifications have been met (Cohen et al., 2007a).

Although desirable, testing all the possible software inputs exhaustively is impracticable owing to a common problem in software testing called combinatorial explosion problem (Cohen et al., 1997; Cohen et al., 1996; Colbourn, 2009; Colbourn, 2011; Tai & Lei, 2002). In real-life, software inputs (parameters and their value dependencies) are typically very large. For this reason, the configurations for testing consideration are exponentially expanding with the increased number of software inputs. Interaction testing (*t*-way) strategy has been known to successfully reduce the test cases for testing consideration. However, generating the minimum *t*-way test set is challenging because of the potentially large search space. Furthermore, such a problem is also considered as NP-Complete (i.e. Non-deterministic Polynomial-time) problem (Petke, 2015).

In the last 10 years, researchers have advocated the adoption of meta-heuristic based *t*-way strategies in line with the emergence of the new field called Search based Software Engineering (SBSE). To date, many *t*-way strategies have been proposed in the literature such as Genetic Algorithm (GA) (Chen & Chien, 2011; Lopez-Herrejon et al., 2016; McCaffrey, 2009a, 2010; Sabharwal et al., 2016; Shiba et al., 2004; Srinivas & Deb, 1994), Simulated Bee Colony (SBC) (McCaffrey, 2009b), Ant Colony Optimization (ACO) (Chen & Chien, 2011; Chen & Zhang, 2009; Shiba et al., 2004), Simulated Annealing (SA) (Chen & Chien, 2011; Cohen et al., 2008a; Cohen et al., 2007b; Stardom, 2001), Particle Swarm Optimization (PSO) (Ahmed & Zamli, 2010a; Ahmed & Zamli, 2010b; Ahmed & Zamli, 2011b), Harmony Search (HS) (Alsewari & Zamli, 2012a), Hill Climbing (HC) (Alsewari et al., 2014; Zamli et al., 2015) and Cuckoo Search (CS) (Ahmed et al., 2015) to mention a few. Further elaboration on these strategies can be seen in Section 2.5.

Although helpful, no single meta-heuristic based *t*-way strategies can claim dominance over their other counterparts. For this reason, the search for a new meta-heuristic based *t*-way strategy is still a useful endeavour. Bat-inspired algorithm (BA) is one of many newly developed meta-heuristic algorithms in the literature. Taking into account on its superiority over other meta-heuristic algorithms (i.e. GA, SA, HS and PSO) (Sureja, 2012), the performance of BA can be seen throughout several studies on optimization problems (Gherbi et al., 2014; Hegazy et al., 2015; Khan & Sahai, 2012; Senthilnath et al., 2016; Sureja, 2012; Taha et al., 2013; Yang, 2010) (Ali, 2014; Gandomi et al., 2013; Meng et al., 2015; Nguyen & Ho, 2016; Rakesh et al., 2013; Ramesh et al., 2013; Rodrigues et al., 2014; Song et al., 2016). Despite its performance owing to superior exploration (i.e. manipulating in the region close to the best solution so far), existing BA appears to suffer from lack of diversification (i.e. exploring the solution space at the global scale). Therefore, the Hamming distance classifier is selected to enhance the exploration of BA as this method improves the selection of best test candidate for *t*-way test suite generation problem (Gonzalez-Hernandez, 2015). Ideally, the Hamming distance classifier ensures that the highest distance solution in the search space is selected when there are ties as far as the best candidates of test cases are concerned. Further elaboration on the superiority of BA and Hamming distance will be elaborated at the end of Chapter 2.

The fundamental research questions are:

RQ 1. What is the optimum *t*-way and mixed-strength test suite (i.e. smallest number of test cases) to be considered for testing?

RQ 2. How effective can the *t*-way strategy perform the sampling from a large combinatorial test data?

RQ 3. Will the optimum *t*-way test suite generated based on the BA algorithm effectively cover all test configurations to detect interaction bugs?

Given the aforementioned prospects, this thesis presents the design and implementation of a new meta-heuristic based *t*-way strategy, called Bat-inspired *t*-way Strategy (BTS), for generating mixed-strength *t*-way test suite. BTS is the first *t*-way strategy that adopts BA as its core implementation and exploits the Hamming distance as the final selection criteria. It is the hypothesis that suggests the adoption of BA is useful for generating optimum mixed-strength *t*-way test suite is the main focus of this work.

1.4 Research Aim and Objectives

The aim of this research effort is to propose a new *t*-way test generation strategy that supports mixed-strength interaction, called Bat-inspired *t*-way Strategy (BTS), augmented with Hamming distance classifier. Supporting the aim, the objectives of the research are:

i. To study the design of BTS strategy for constructing a mixed-strength *t*-way test suite.
ii. To model BTS as a research prototype using BA as the backbone search engine and introduces Hamming distance classifier in order to enhance the exploration of BA.
iii. To evaluate the test suite size performance of BTS against existing strategies using well-known benchmarking case studies.

1.5 Research Scope

This research work focuses on the test case generation stage in the software testing life cycle. Specifically, the research work is to address the mixed-strength *t*-way sampling/generation for test execution.

The scope of this research work is limited to the implementation of a *t*-way test generation strategy, BTS, taking the Bat algorithm as the core implementation. The current support interaction strength is set at $t = 6$ consistent with empirical evidence (where most (if not all interaction bugs) can all be detected).

The focus of the work is on test planning (i.e. test generation) and not on test execution. As such, the performance of BTS for mixed-strength *t*-way test generation is based on its optimality (in terms of getting the most minimum test suite size).

1.6 Operational Framework

The complete research operational framework used throughout this research work is illustrated in Figure 1.4. Here, the operational framework is divided into three main stages: literature review, research methodology, and the evaluation stage for the proposed strategy.

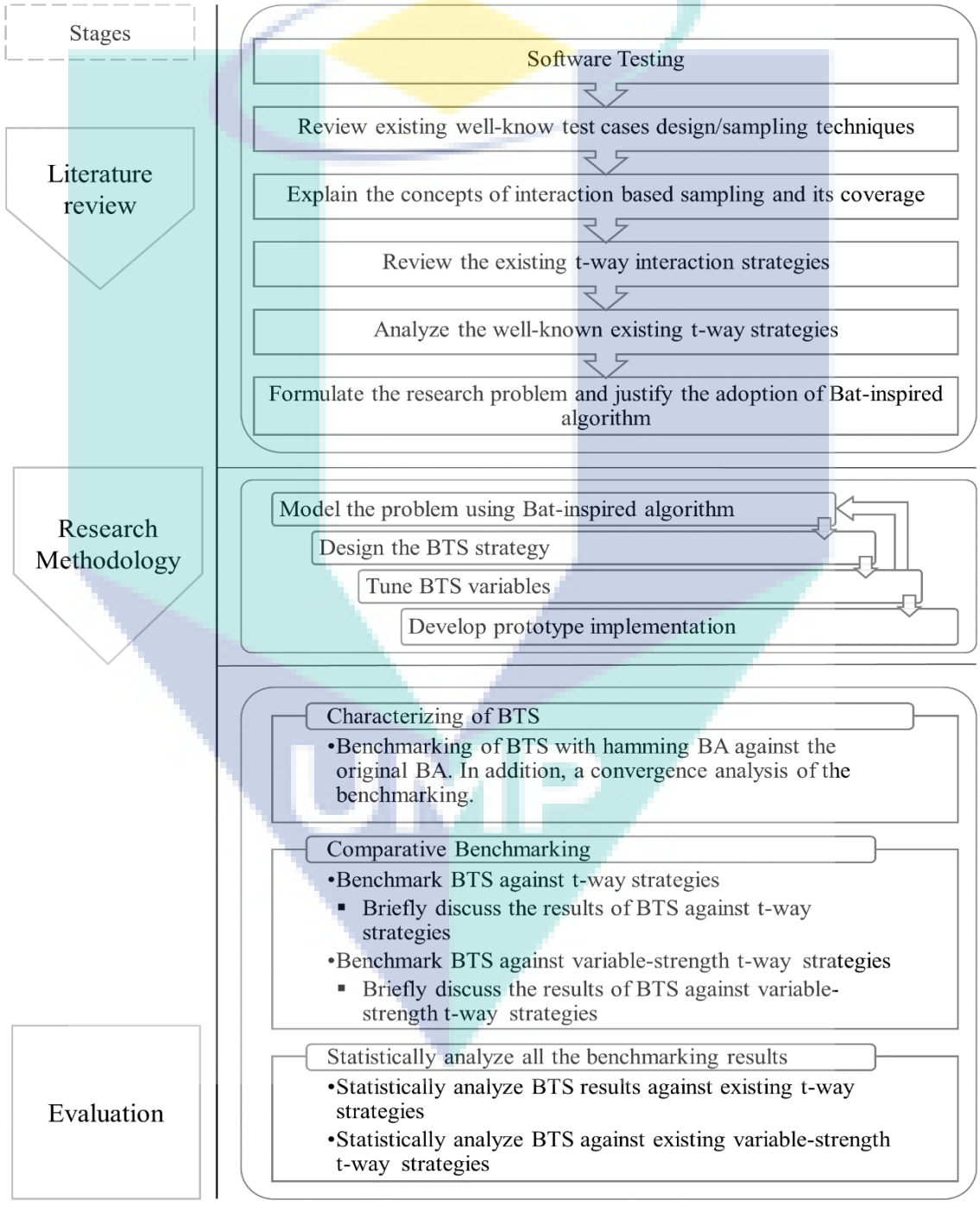


Figure 1.4 The illustration of the operational framework for BTS.

The illustrated stages are elaborated to show how the stages are related as follows:

Literature review stage involves identifying the core problem on combinatorial explosion of test cases. Based on the core problem, the work reviews how the combinatorial problem is currently being addressed in the literature. Based on the literature search, state-of-art sampling strategies including *t*-way interaction testing strategies are reviewed whereby the research gap is established in terms of the adoption of BA. From that, the requirement of the research is established to provide the justification for the adoption of BA.

Research methodology (design and implementation) stage involves finding the best model for BA implementation. During this stage, it is decided that BTS strategy will be developed based on "one-test-at-a-time" approach in order to achieve best size performance. Then, the complete algorithms constructing the BTS strategy are designed and developed. BTS prototype is be implemented using Java programming language, in order to support cross-platforms environment (i.e., Mac, Linux, and Windows operating system).

Finally, evaluation stage involves three main sub-stages: characterization of BTS, comparative benchmarking and statistical analysis respectively. Characterization and comparative benchmarking will be performed based on well-known configurations in the literature (refer to Chapter 4). By using well-known configurations, more objective comparison can be made amongst different strategies of interests.

1.7 Thesis Organization

The remainder of this thesis is organised into six chapters. The current chapter gives an overview of software testing including the basics of test case generation. Then, the *t*-way test generation strategy has been introduced in line with the new field of Search Based Software Engineering (SBSE). Finally, the problem statements as well as the research questions and aim are highlighted.

Chapter 2 presents an overview of test case design (or *sampling*) techniques. Then, an overview of the mathematical notation used in interaction test generation in order to elaborate the concept of *t*-way interaction test generation based on a defined problem (running example) as well as highlighting the main characteristics of *t*-way interaction test generation. Using the characteristics, a survey of existing *t*-way test generation strategies is provided including a special case for *t*-way that is mixed-strength interaction that corresponds to a sub-strength generation. Towards the end of the existing *t*-way survey in Chapter 2, an analysis of existing work is presented, which provides the requirements and justification for the development of BTS.

Chapter 3 discusses and justifies the detailed *t*-way test generation design and implementation for BTS. Here, issues related to the enabling automated test generation are also explained. Additionally, the prototype implementation is also discussed in order to highlight its usage.

In Chapter 4, a detailed account for evaluating BTS is presented. Here, the performance of the BTS strategy will be evaluated. Apart from the performance evaluation, a comparative study on the effectiveness of test suite generation will be highlighted using several real-world software test configurations. Additionally, BTS will also be compared against existing strategies in terms of the number of generated test cases for *t*-way and mixed-strength test suite generation.

Finally, the conclusion of this work is given in Chapter 5, where the achievements and contributions are summarised. Additionally, the main research hypothesis is revisited and the usefulness of BTS is debated. Conclusions are drawn from the experience gained from this work and the significance of findings along with considerations for future work.

CHAPTER 2

LITERATURE REVIEW

Common test generation problems have been briefly introduced in Chapter 1 leading towards t-way interaction strategies. Owing to its importance, this chapter elaborates further on these well-known test case design strategies. Then, a review of the necessary mathematical notation used for t-way testing is introduced along with the notation for mixed interaction. Next, an overview of t-way interaction testing, including the problem of t-way interaction is identified using a mixed interaction running example along with the interaction tuples coverage mechanism. Towards the end, an analysis and review of the literature follows the discussion. Finally, the advantages of BA are detailed out to justify its adoption for BTS implementation.

2.1 The Test Case Design Strategies

Software testing is considered as a planned activity within the software development life cycle. Before the execution of any test, test engineers need to prepare the appropriate test suites based on some sampling strategies (as exhaustive testing is impossible). In the next sub-sections, the well-known test case design strategies are reviewed highlighting their focus and importance.

2.1.1 Random Testing

Random testing (Duran & Ntafos, 1984) is one of the first test case design strategies used in software testing. Random testing as the name suggest is trying to generate random test cases. In some random testing approaches, invalid or unexpected inputs are randomly selected to reveal the defects. Systematic random testing method (Antony, 2003; Schroeder et al., 2004; Tseng et al., 2001), uses a probability sampling in which test cases are selected from the test space using a random staring test case. Then,

the rest of the test cases selected based on a sampling interval. The sampling interval is calculated by dividing the test space size by the needed test cases number. In general, random testing technique (i.e. systematic random testing) is not effective to use as a test suite design (or generation) because of the unfair distribution of test cases (Ammann & Offutt, 1994).

2.1.2 Equivalence Class Partitioning

The equivalence partitioning is used to design test cases for the well-defined inputs and outputs of software-under-test (Burnstein, 2006). Here, the software inputs are partitioned into classes that get equivalent treatment. A test case is selected for each class, considering that all the members of the represented class are treated equivalently by the software-under-test (Hass, 2008). In such a strategy, equivalence partitioning selects the test case from each equivalence class. If the selected test case from the equivalence class reveals a defect, then, the other test cases in the same class should reveal the same defect (as the test case for the class supposed to be equivalent to any other test case in that class in theory) (Sharma & Chandra, 2010). Similarly, if the selected test case does not reveal any defect, the other test cases in the same class should not reveal any defect, in other words, no further execution for specified class is required. Moreover, if a test case in a class revealed any defect, the others test data in the same class should not reveal any defect or vice versa, the defined equivalence classes are considered not correct or valid for the test execution. In this case, the equivalence classes are re-defined or divided into smaller classes. However, there is no defined role for selection the values in the equivalence class. Usually, the values are selected randomly based on the range and condition for the test parameters. Figure 2.1 illustrates a simple grade converter software to clarify this technique clearly.

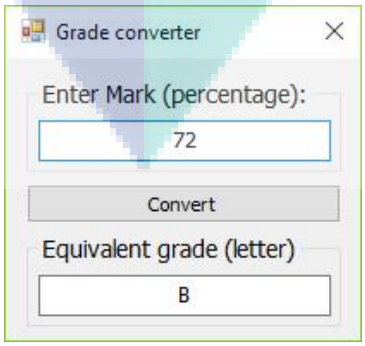


Figure 2.1 A grade converter software.

The simple converter software in Figure 2.1 views the corresponding grade letter for the score entered. The letter depends on the entered value from 0 to 100 as shown in Table 2.1.

Table 2.1 The grading and their corresponding letters.

Score	Grade
80 -100	A
70 - 79	B
60 – 69	C
50 - 59	D
0 - 49	F

As the input value for the score is a numeric value, there are infinite values that might be tested, in this case, five valid partitions might be selected as follows; 80 to 100, 70 to 79, 60 to 69, 50 to 59 and 0 to 49. Here, the valid partitions also might be to split into more than one partitions as 0 to 49 could also be further divided into 0 to 25 and 26 to 49. Then, a test case or value is selected from each of the partitions (i.e. 15, 47, 55, 62, 74 and 85) is selected to be tested. Furthermore, other invalid equivalent class partitions should be defined; one with all the values less than 0 and the second invalid class with the values over 100. Thus, the test cases for this software based on this technique might be -5, 15, 47, 55, 62, 74, 85 and 120. However, defects might be occurring at the edges of equivalence classes (Myers et al., 2011).

2.1.3 Boundary Value Analysis

Boundary value analysis is a test case design technique similar to the equivalence class partitioning. The test cases in the boundary value analysis are selected similarly to the equivalence class technique with the present of the representatives of the boundary values of the edge of equivalence classes as many defects might be occurred on the edges of equivalence classes (Burnstein, 2006; Myers et al., 2011). Furthermore, the input space and the output space are being considered in the boundary value analysis. Like equivalence class partitioning, boundary value analysis has no specific method to achieve the best test suite design. The boundary value analysis mostly depends on the creativity of the test engineers to achieve the best test suite for the targeted software-under-test.

In such a technique, the range of values is determined. Then, the valid values or test cases are selected in the edge of each range. Furthermore, invalid test cases are selected beyond the boundaries of the test space. For example, a system has a range values from 0 to 100. In this situation, the boundary values 0 and 100 are a valid input to consider as test cases. Then, two invalid values beyond the selected boundary values are considered. In this situation, -1 and 101 are selected. Revisiting the grade converter software in Figure 2.1, the ranges (see Table 2.1) are corresponding to a different letter; each range is treated similarly as in the example mentioned above. Therefore, the test engineers must select the upper and the lower value for each boundary value. Thus, the test cases should be (-1, 0, 1, 48, 49, 50) for the output grade "A." Here, the test suite for the grade converter should be (-1, 0, 1, 48, 49, 50, 51, 58, 59, 60, 61, 68, 69, 70, 71, 78, 79, 80, 81, 99, 100, 101).

2.1.4 Cause and Effect Graphing (CEG)

Cause and effect graphing (CEG) (Nursimulu & Probert, 1995) is proposed to validate a software from its requirements specification. In other words, CEG generates test cases by transforming software requirements specification natural language into an acyclic Boolean logic network. This logic network contains two main logical relationships; inputs (*causes*), outputs (*effects*). Then, the constraints among the causes or effects are represented using a limited-entry decision table. Here, each column within the decision table represents a test case (Myers et al., 2011; Naik & Tripathy, 2008),

In this technique, test engineer identifies the causes, effects, and limitations (*constraints*) from the requirements specification of the software-under-test. Then, an acyclic Boolean logic network is constructed, that graphically represents the identified causes, effects as nodes and their constraints. These nodes of causes and effects are connected with Boolean operators (not, or, and). Next, an identifier for each cause and effect is assigned. Then, the relationship between causes and effects is assigned. Finally, the cause and effect graph is transformed into a decision table.

For the abovementioned example in Figure 2.1, assuming that the requirements specification of the "Grade converter" software, the "Mark" field should only accept numbers and its length should not exceed three digits. In such a case, the software shows an error message "out-of-range." Otherwise, the equivalent grade will appear. Hence, the range of the "Mark" is from 0 to 100. Here, the inputs or causes are as follows; C1: The mark is from 0 to 49, C2: The mark is from 50 to 59, C3: The mark is from 60 to 69, C4: The mark is from 70 to 79, and C5: The mark is from 80 to 100 (refer to Table 2.1), C6: The mark is out-of-range. Whereas, the output or effects are E1: The equivalent grade is "F," E2: The equivalent grade is "D," E3: The equivalent grade is "C," E4: The equivalent grade is "B," E5: The equivalent grade is "A," respectively. As for, E6: The input is an out-of-range. Figure 2.2 illustrates the identified causes and effects. In addition, their relationships.

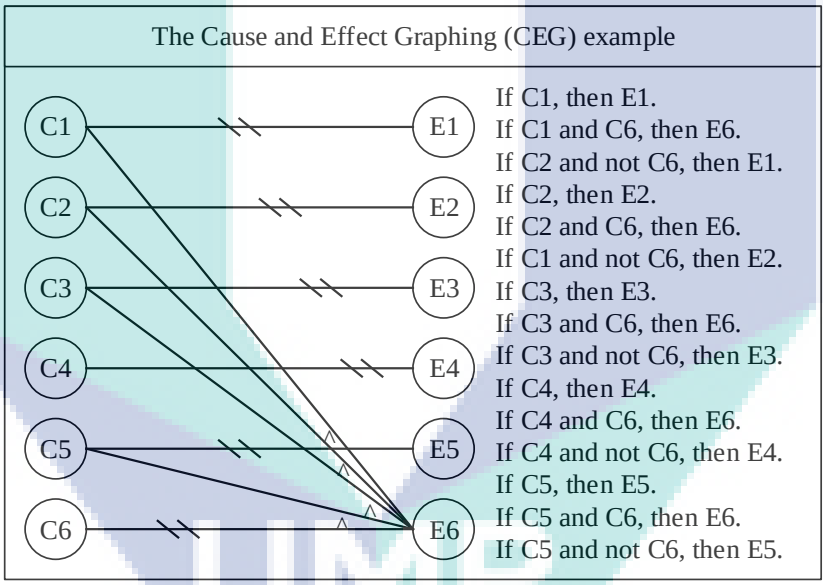


Figure 2.2 The CEG for the Example in Figure 2.1.

The next step is to construct the decision table based on the causes and effects. The first column contains the case based on the causes and effects, then each cell in the same row field with it corresponding value, the values can as follows; "1" indicates the inclusion of the reparative case, "0" indicates the exclusion reparative case and "*" indicates a "don't care" value. Here, Table 2.2 shows the decision table for Figure 2.2. The columns TC1, TC2, TC3 represent the test cases that could be used to test the "Grade Converter" software.

Table 2.2 The decision table for the CEG in Figure 2-2.

Case	TC1	TC2	TC3
C1	1	0	0
C2	0	1	0
C5	0	0	1
C4	0	0	0
C5	0	0	0
C6	1	1	0
E1	1	1	*
E2	0	0	*
E3	0	0	0
E4	0	0	0
E5	0	0	0
E6	1	0	1

2.1.5 Interaction Sampling

Complementing the strategies highlighted earlier (i.e. systematic random testing, equivalence class partitioning, boundary value analysis, and cause and effect graphing), interaction *t*-way strategy deals with the interaction between the software inputs as faults could be triggered owing to the interaction itself. The next section and its subsection elaborate on the mathematical notations as well as fundamental concepts for *t*-way test suite generation.

2.2 The Mathematical Notations for *t*-way Test Suite Generation

Empirical research results indicate that software systems failure is mostly triggered by interactions among t parameters from $(2 \leq t \leq 6)$ (Czarnecki et al., 2012; Kuhn et al., 2004). Interaction testing is a method capable of constructing the test suite that covers all *t*-way parameter-values. Commonly, the mathematical notation for interaction testing including *t*-way test generation is driven from algebraic mathematical properties of inputs and its values based on the Covering Array (CA) and Mixed-level Covering Array (MCA) notations (Cohen, 2004).

The notation CA has four main parameters, namely, S, t, p, and v (i.e. CA (S, t, P, v)). CA is a matrix of size $S \times P$. Here, The symbols t refers to the interaction strength, S represents the test cases (rows), P is known as number of parameters (columns) and v refers to the number of CA values for a specific P (Kuliamin & Petukhov, 2011; Yilmaz et al., 2006).

CA $(S, 2, 4, 3\ 3\ 3\ 3)$ is equivalent to CA (S, t, v^P). Hence, for CA $(S, 2, 3^4)$, it can be seen as $S \times 4$ array that covers the test suite. In this case, the test suite covers t = 2 (or termed *pairwise interaction strength*) with three v values and four p parameters, S is $3 \times 3 = 9$ test cases. This case represents the most minimum covering array S (see Figure 2.3 (A)) as shown is Equation 2.1 based on the definition of CAN (Hartman & Raskin, 2004b).

$$optimal\ CA \rightarrow CAN\ (t, v^P) = \min\ \{\exists | \lambda\ CA\ (S, t, P, v)\} \quad 2.1$$

Nonetheless, systems usually have different values for each of its components. Therefore, MCA is introduced to represent each component as individual test parameter as its own values. MCA is indicated by MCA $(S, t, p, v_1 v_2, \ldots, v_n)$. The only difference to that of CA is that v_n is specified values for related P parameter for every single column $v_i \in (1 \leq i \leq P)$ containing elements of the set $|v_n^*| = v_n$. The row of each submatrix contains $S \times t$ interaction elements that cover all t-tuples from the related t columns at least once. Similar to CA, MCA can be indicated as MCA $(S, t, P, v_1^{x1} v_1^{x2}, \ldots, v_n^{xi})$, where p as shown is Equation 2.2.

$$P = \sum_{i=1}^{n} x_{\mathrm{n}} \quad 2.2$$

In here, each x_i parameter has its own v_n value, for example, a test suite covers pairwise interaction strength with four P parameters, first two parameters have two values and the rest two has three values. MCA is formulated as MCA $(S, 2, 2^2\ 3^2)$. In case of the most optimal value of S, MCA is represented as in Equation 2.3 (see Figure 2.3 B) for the most optimal test suite.

$$\text{MCAN}\left(t, v_1^{P_1}\ v_2^{P_2}, \ldots, v_n^{P_i}\right) = min\ \{\ \exists\ |\ \lambda\ \text{MCA}\ (S, t, P, v_1, v_2, \ldots, v_n)\} \quad 2.3$$

In order to clarify the use of a CA and MCA for interaction testing, the reader is referred the representation of the most optimal test suite for the given example in Figure 2.3 (B).

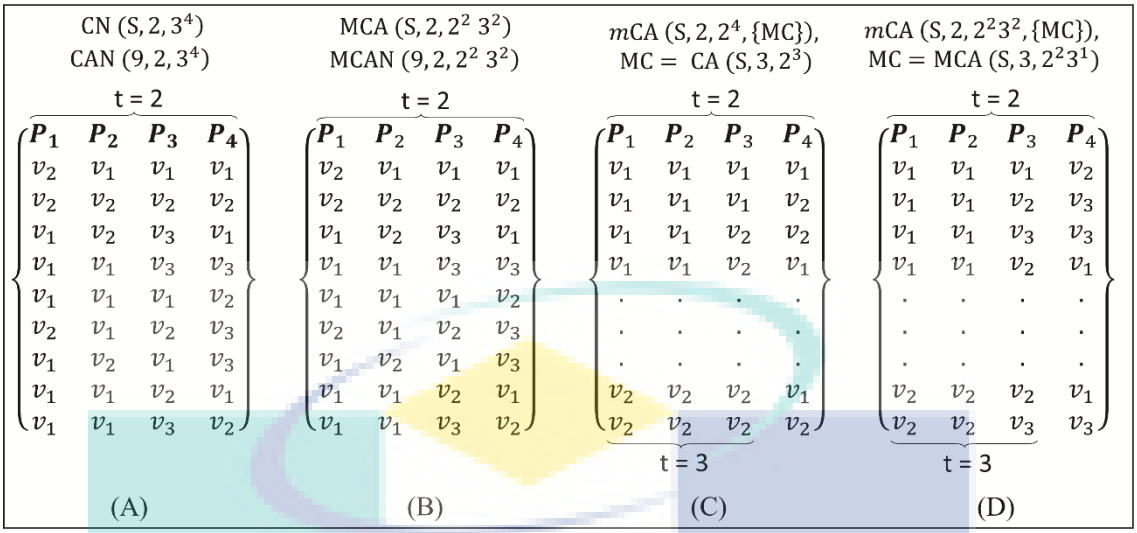


Figure 2.3 The illustration of the mathematical notations.

As this study considered mixed-strength interaction (as variable-strength interaction only) (Bansal et al., 2015; Cohen et al., 2003a), the mixed-strength Covering Array (mCA) and mixed-strength Mixed-level Covering Array (mMCA) are following the same concepts of a CA and MCA with added condition for the mixed-strength interaction (MC). Referred to as mixed-strength interaction condition, the mCA is formulated as mCA $(S, t, v^P, \{\text{MC}\})$ and mMCA $\left(S, t, v_1^{P_1}\, v_2^{P_2}, \ldots, v_n^{P_i}, \{\text{MC}\}\right)$, respectively, where the condition (MC) is one or more CA or MCA.

For instance, mCA $(S, 2, 2^4, \{\text{MC}\})$, where MC = CA $(S, 3, 2^3)$ (see Figure 2.3 (C)) implies a test size of S for t = 2 for all four parameters along with a sub-strength t = 3 for the first three parameters. The interaction elements for the main configuration in the mCA and the sub-configuration CA (mixed-strength) are combined together to generate a test suite of size S. The running example in problem definition model section gives further description for the $mMCA$ covering array shown in Figure 2.3 (D).

2.3 A Problem Definition Model for *t*-way Test Suite Generation

In order to illustrate how the *t*-way test generation works, consider the source code inputs in Figure 2.4 as a running example. For simplicity, we consider a source code with two Boolean inputs and two arrays each of which has three characters. Here, the input consists of four parameters; P_1, P_2, P_3, and P_4. P_1 and P_2 have two values each (i.e. true or false) referred to as, $P_i v_1$ and $P_i v_2$, respectively. On the other hands, P_3 and P_4 have three values each (i.e. P_3 = {'<', '=', '>'} relation values, P_4 = {'+', '-', '*'} operation values) referred to as, $P_i v_1$, $P_i v_2$ and $P_i v_3$, respectively.

From this point on, the use of "combination" reflects the relation between the domain-under-test parameters as a set of parameters (i.e. x = { P_1 , P_2 }) (i.e. the combination of P_1 and P_2 and without their dependency values). The set of parameters-values (i.e. x = {$P_1 v_1$, $P_2 v_1$, $P_3 v_1$, $P_4 v_1$}), is referred to as "configuration" as a set of the parameter dependencies values. Here, the use of the parameter (P_i) refers to which parameter the value (i.e. v_1, v_2 *or* v_3) is related.

Running Example

```
boolean P1, P2;
char [] P3 = {'<','=','>'};
char [] P4 = {'+','-','*'};
```

Input Parameters

	P_1	P_2	P_3	P_4
Values	True	True	<	+
	False	False	=	-
			>	*

Figure 2.4 Running example.

Figure 2.4 demonstrates the input parameters, as a set of four parameters. Assuming that each parameter is triggering a specific action, the system can be tested based on the four parameters; P_1 , P_2 , P_3 and P_4 , respectively. Each of these four parameters has its own values (i.e. P_1 = {$P_1 v_1$, $P_1 v_2$}, P_2 = {$P_2 v_1$, $P_2 v_2$}, P_3 = {$P_3 v_1$, $P_3 v_2$, $P_3 v_3$} and P_4 = {$P_4 v_1$, $P_4 v_2$, $P_4 v_3$}) as seen in Figure 2.5.

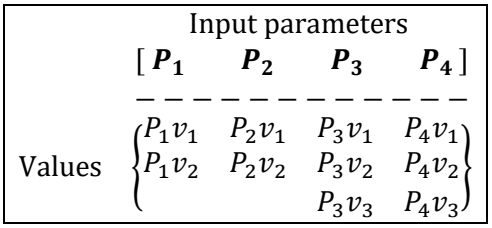


Figure 2.5 The running example parameters and values.

The maximum number of test configurations (exhaustive configurations at t=4) can be calculated based on the Equation 2.4. Where $P_{\text{set } i}$ and $v_{\text{set } i}$ are sets of similar parameters and values. Thus, the exhaustive configurations consist of $2^2 \times 3^2 = 36$ test configurations (or *cases*) as shown in Figure 2.6.

$$\textit{Number of configurations} = (v_{\text{set } 1})^{P_{\text{set } 1}} \times (v_{\text{set } 2})^{P_{\text{set } 2}} \times, \ldots, (v_{\text{set } i})^{P_{\text{set } i}} \qquad 2.4$$

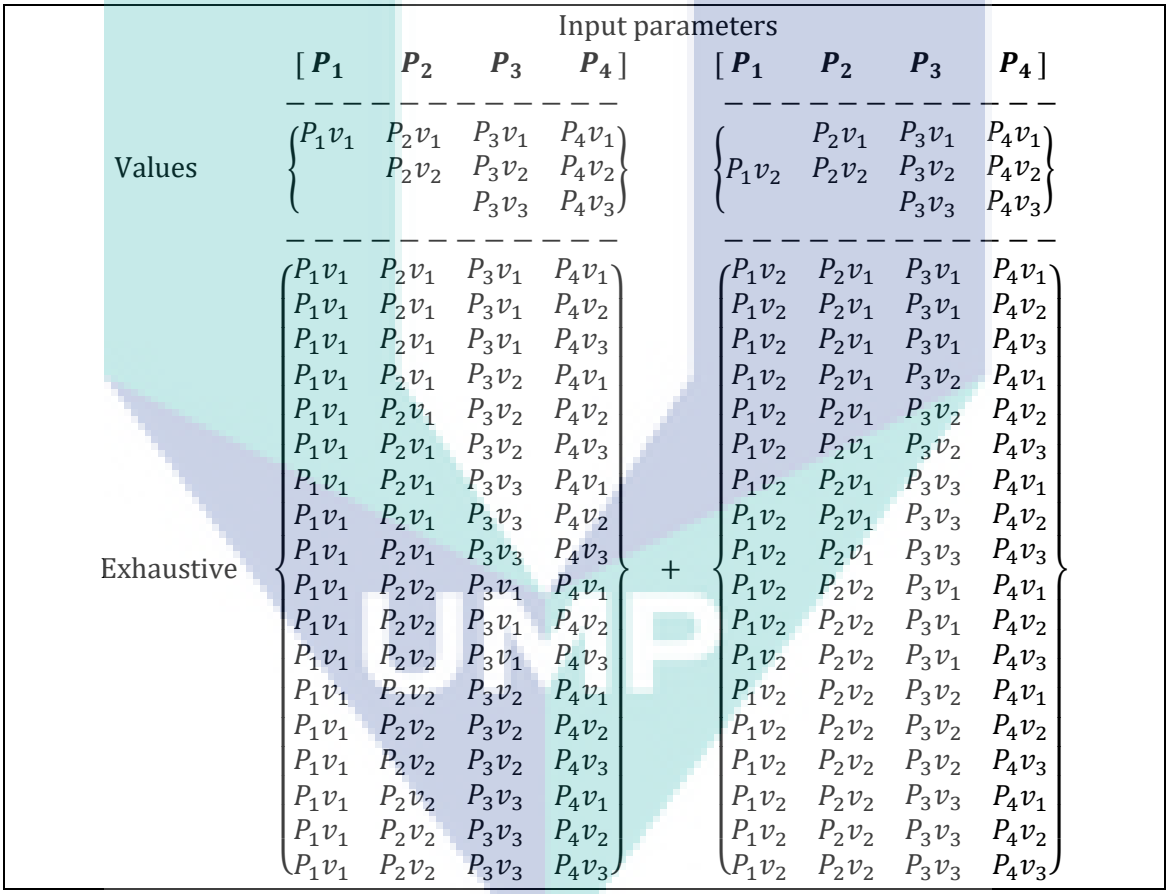


Figure 2.6 The exhaustive test suite (at t = 4) for CA(36,4, 2^2 3^2).

A closer look in Figure 2.6, the parameters can be viewed as columns in a matrix; P_1, P_2, P_3 and P_4, respectively. For column P_1, the value P_1v_1 is repeated 18 times then P_1v_2 is also 18 times to reach the maximum number of test cases (in Figure 2.6, P_1v_1 and P_1v_2 are separated for the sake of illustration simplicity), which gives 36 cases in total. Here, the value repeated 18 times, because there are 36 configurations with 2 specified values for each. Therefore, 36 configurations are divided by 2 as specified by the repetition for P_1(18 times). Then, for the column P_2, the value 18 is also divided by the specified values for P_2 which is 2 (9 times). The value of P_2v_1 is repeated 9 times then P_2v_2 is also 9 times, Here, both of P_2v_1 and P_2v_2 are repeated 9 times each. In similar manner, for column P_3, the values; P_3v_1, P_3v_2 and P_3v_3 are repeated alternately 3 times each. For P_2 = 9, the repetition is P_3 divided by 3. Finally, for column P_4, P_4v_1, $P_4v_2$and $P_4v_3$are repeated alternately until 36 configurations is reached.

Here, to minimize the exhaustive test suite from full interaction strength (t = 4) in Figure 2.6, consider relaxing the interaction strength to 2-way (or t = 2) for the first two parameters P_1, P_2. Here, the range of values for (t) is between two (i.e. *2*-way) and exhaustive case (the maximum number of defined parameters). In this case, the parameters P_3, P_4 values could be treated as "don't care", such that the values of P_3, P_4 could be randomly assigned (i.e. P_3v_1, P_3v_2 or P_3v_3 for P_3 and P_4v_1, P_4v_2 or P_4v_3 for P_4). Then, the number of test configurations for 2-way can be calculated based on the Equation 2.5 yielding $2^2 = 4$ test configurations. For instance, the configurations follows the set role $\{P_1, P_2\}$ for all the values (i.e. $\{P_1v_1, P_2v_1\}$, $\{P_1v_1, P_2v_2\}$,$\{P_1v_2, P_2v_1\}$, $\{P_1v_2, P_2v_2\}$) (see Figure 2.7 (A), where, "x" refers to "don't care" values). Then, P_3, P_4 values are added randomly. Using this technique, the number of test configurations for 2-way test suite can be minimized to 4 test cases (see Figure 2.7 (B)).

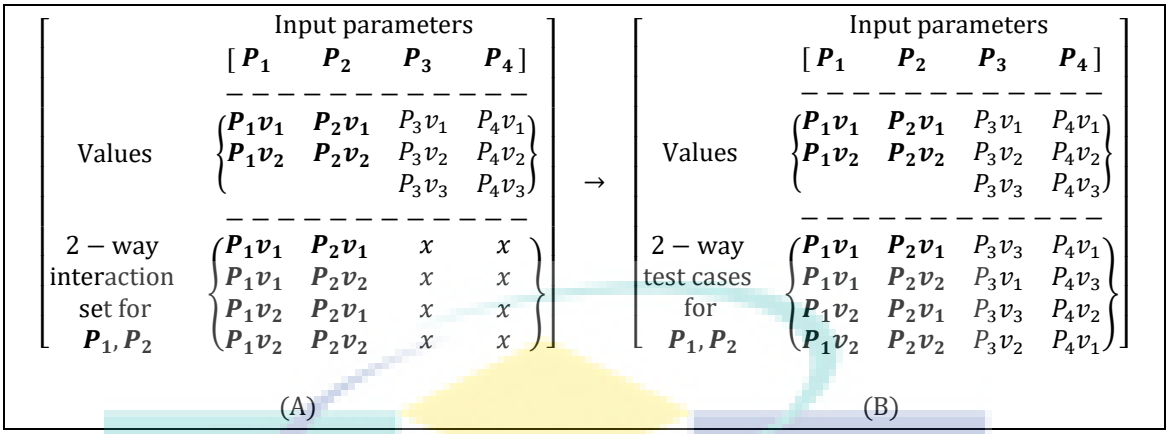


Figure 2.7 The illustration of the interaction elements set in (A) and the test cases set in (B) for 2-way interaction strength (at t = 2).

In real world software, it is difficult to point out which software parameter has insignificant effect or impact on the software. As a matter of fact, considering the impact of other than 2-way combinations may be needed. Therefore, the *t*-way test suite must be generated for all the software parameters to cover all (*t*) interaction for the software. Thus, all the possible parameters combinations for 2-way (i.e. (P_1, P_2), (P_1, P_3), (P_1, P_4), (P_2, P_3), (P_2, P_4), and (P_3, P_4)) need to be considered. In this example, there are six possibilities for 2-way interactions combinations need to be considered using similar method as illustrated for the P_1, P_2 (see Figure 2.7). Here, there are six sets of combinations for 2-way interaction configurations, these configurations can be configured based on the six sets of combinations as follows; set i = {(P_1, P_2, x, x}, set ii = {P_1, x, P_3, x}, set iii = {P_1, x, x, P_4}, set iv = {x, P_2, P_3, x} , set v = {x, P_2, x, P_4} and set iv = {x, x, P_3, P_4}. In here, "x" can be randomized based on the values of the represented parameters.

Essentially, the interaction configurations with the do not care value "*x*" are called "interaction elements" (see in Figure 2.8 (A)). Typically, these interaction elements are growing exponentially when the number of the parameters and their values increases. The number of interaction configurations can be calculated using the following Equation (see Equation 2.5 (Colbourn & Dinitz, 2006).

$$Interaction\ elements\ size\ = \binom{P}{t} v^t = \frac{P!}{t!\,(P-t)!}\, v^t \qquad 2.5$$

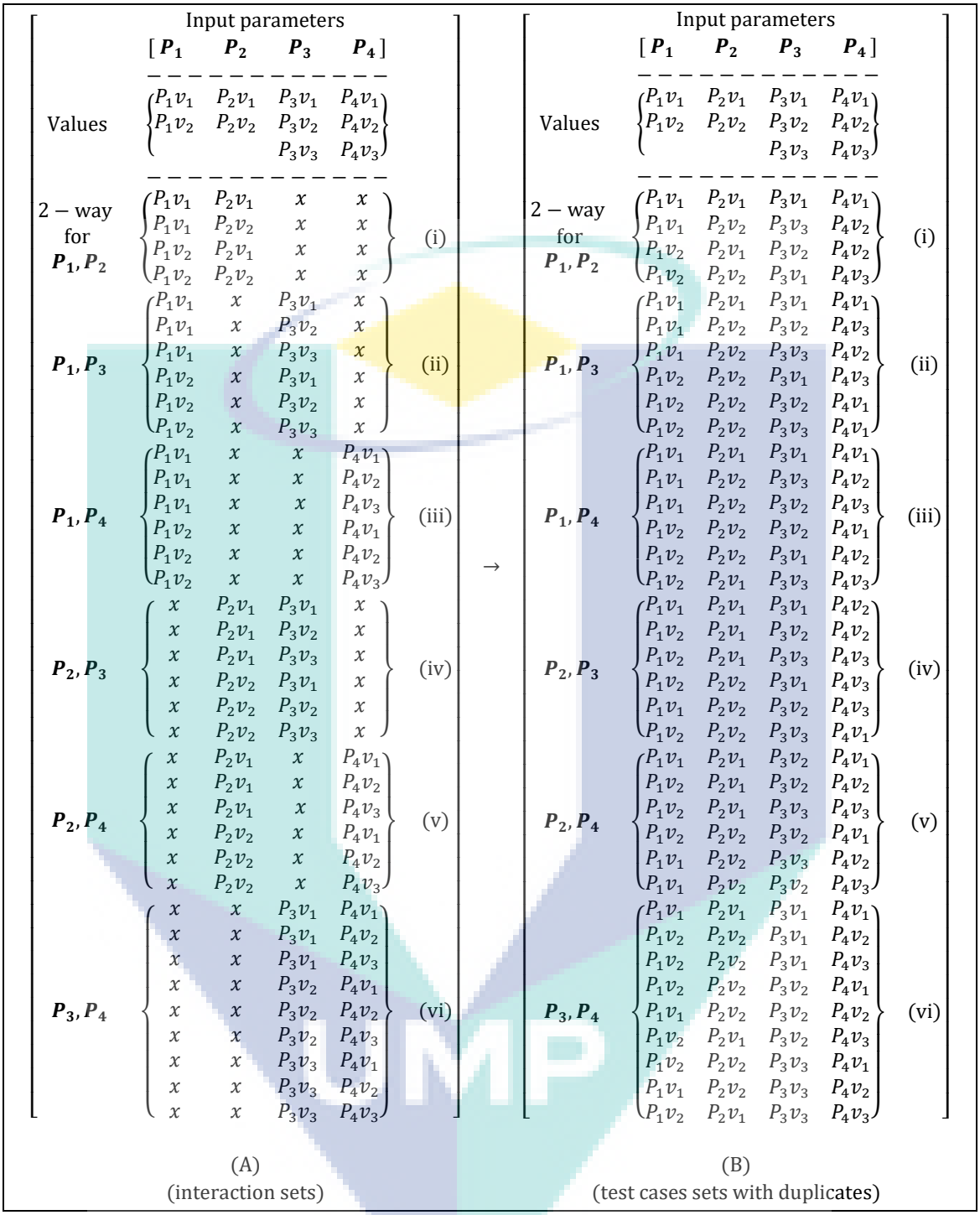


Figure 2.8 The running example interaction elements and test cases sets including randomized values.

To finalize the test suite in Figure 2.8 (B), merging all the 2-way test cases sets can be considered in order to minimize and eliminate the duplicated test cases so as to produce a final test suite in Figure 2.9.

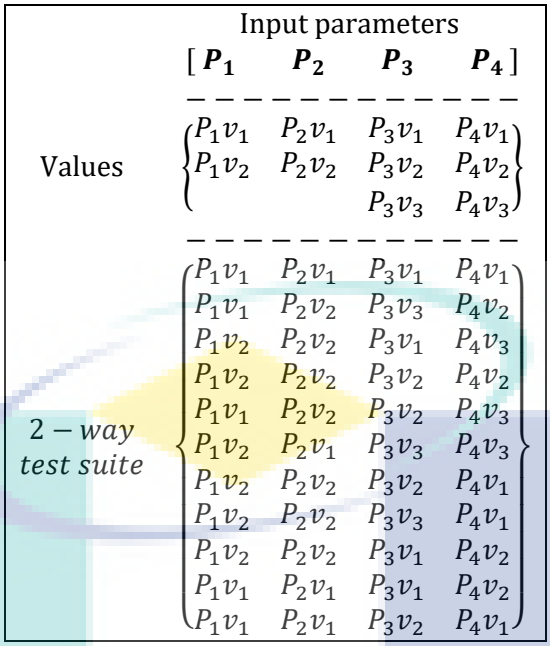


Figure 2.9 Merging of all 2-way test sets, Final test suite for CA(11,4, 2^2 3^2).

Referring to Figure 2.9, it can be noted that the total test suite has been reduced from 36 (at full interaction strength t = 4) to 11 (at t = 2), Here, approximately 70% reduction has been achieved using the "random" based *t*-way *strategy*. From this example, significant test reduction advantage can be observed, which can potentially minimize the test execution cost and time.

To evaluate the final test suite achieved in Figure 2.9, a coverage analysis of the 2-way elements (or interaction tuples) has to be conducted. This example have six combinations elements as follows; (P_1, P_2), (P_1, P_3), (P_1, P_4), (P_2, P_3), (P_2, P_4), and (P_3, P_4). Each of these combinations has its own interaction elements (configurations) that need to be covered at least once by the final test suite. As the strategy described in previous paragraphs has been based on random selection, non-optimum results have been produced as some interactions are covered by more than once. The interaction coverage analysis for 2-way test suite can be seen in Figure 2.10.

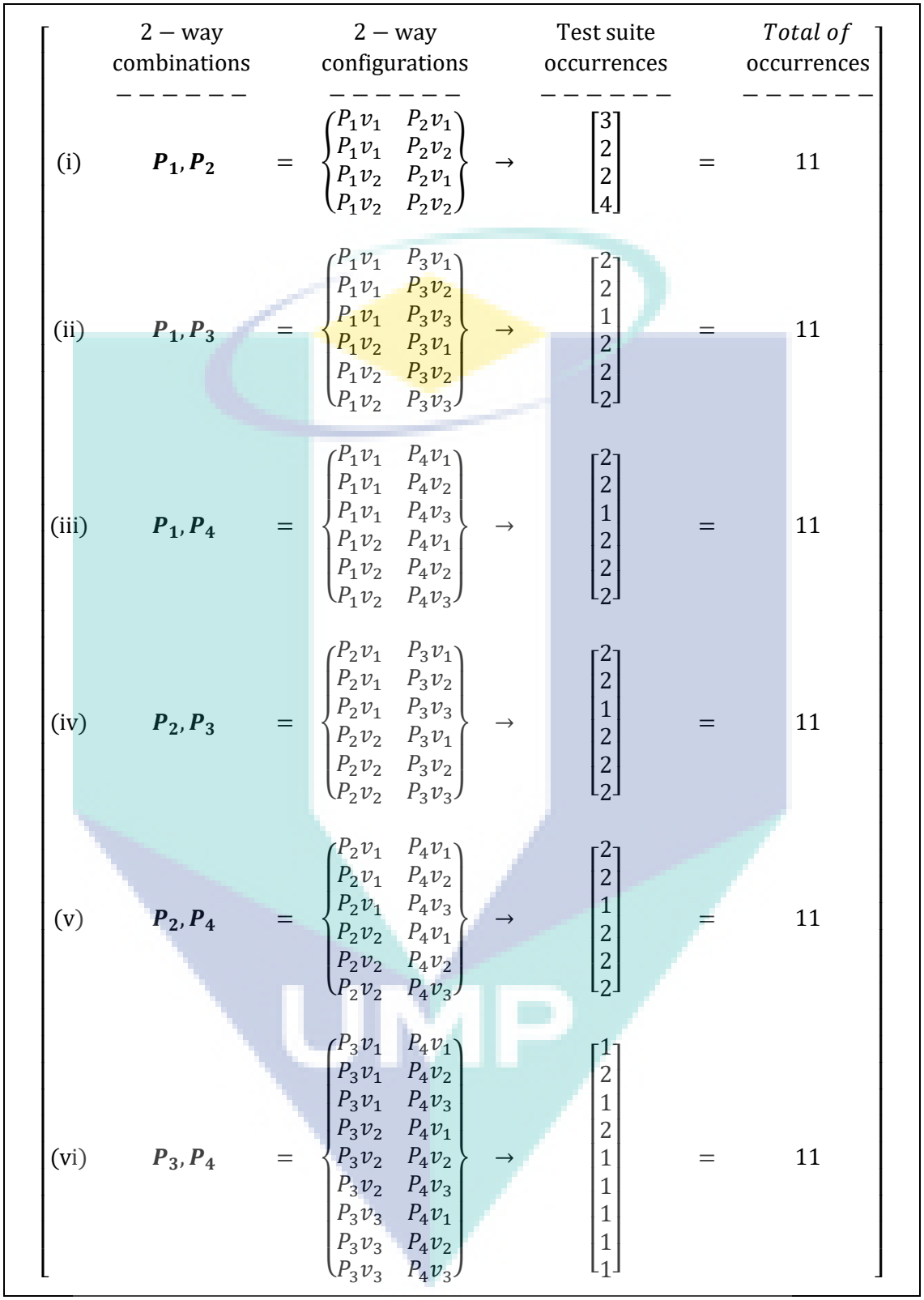


Figure 2.10 Analysis of 2- way interaction configurations occurrence.

Figure 2.10 shows the interaction coverage analysis for 2-way test suite with the test suite achieved for all 2-way interaction configurations (given in Figure 2.8 (A))). The sum of the 2-way interaction configuration occurrences for each configuration must equal

the number total number of test cases in the final test suite, which is 11 (i.e. in Figure 2.10, configuration set i, (P_1v_1, P_2v_1 = 3), (P_1v_1, P_2v_2 = 2), (P_1v_2, P_2v_1 = 2), (P_1v_2, P_2v_2 = 4)), the total of occurrence equals 11 occurrences.

In the case of parameters having a different impact on the software (assuming the defects are caused by the first three parameters), a special case of *t*-way concept called mixed-strength interaction can be considered.

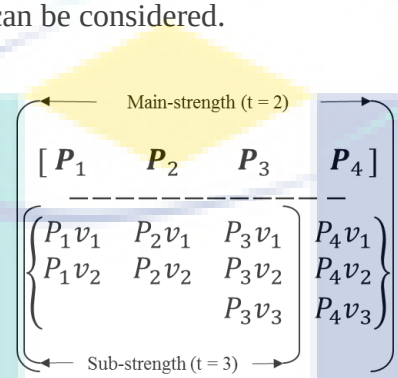


Figure 2.11 The mixed-strength interaction demonstration.

Here in Figure 2.11, assuming that overall defects are caused by all 2-way interactions (*main-strength*), and the 3-way interactions (*sub-strength*) between (P_1, P_2, P_3), the mixed-strength *t*-way interaction can be represented as $mMCA(S, 2, 2^2\ 3^2, CA(S, 3, 2^2\ 3^1))$ whereby the sub-strength CA represents the mixed-strength condition (MC).

The 2-way test suite for the overall (*main*) interaction strength is generated in Figure 2.9. Now, for the 3-way interaction, assuming that P_4 parameter is having insignificant effect on the software. (i.e. "don't care" value)., its parameter can be randomly generated to take any valid values (P_4v_1, P_4v_2 or P_4v_3) each time separately. In effect, for each combination there are only three selected parameters with one randomly generated value. Thus, 3-way interaction test suite is considered by using the set of P_1, P_2, P_3 parameters for t = 3. In this case, the same approach is used to generate the test suite for 2-way for the first two parameters P_1, P_2 as in Figure 2.7 (B. As a result, there are 8 test cases for 3-way interaction strength (see Figure 2.12). In this case, the number of test cases is reduced significantly from 36 to simply 8 test cases for 3-way in this scenario.

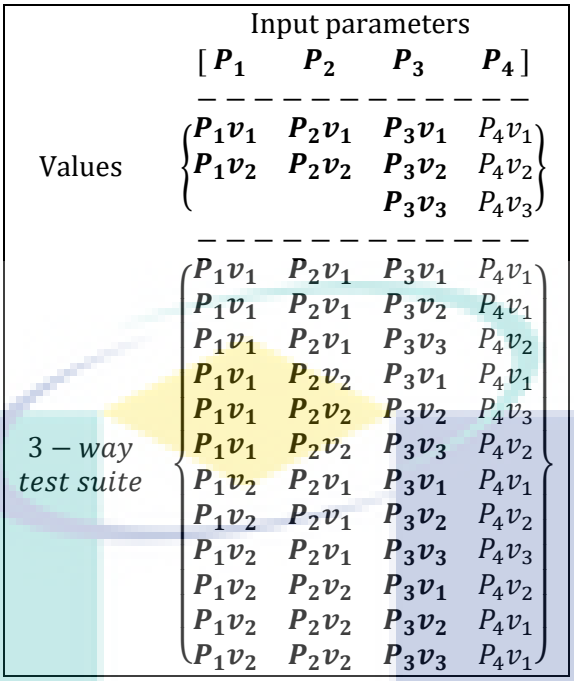


Figure 2.12 The 3-way interaction test suite for the first three parameters.

In order to generate the mixed test suite for this scenario, the 2-way test set for the overall software (see Figure 2.9) is combined with the 3-way test set (sub-strength) for P_1, P_2, P_3 (see Figure 2.12). Then, the duplicates are removed resulting in 15 test cases (see Figure 2.13).

Referring to Figure 2.13, the test cases in Figure 2.13 (B) that are marked in bold text for the 3-way test suite are the ones that are not covered by the overall 2-way test suite. These test cases are added to the ones in Figure 2.13 (A) to construct the final mixed-strength test suite shown in Figure 2.13 (C), the rest (of test cases) in Figure 2.13 (B) are the duplicates that have been eliminated.

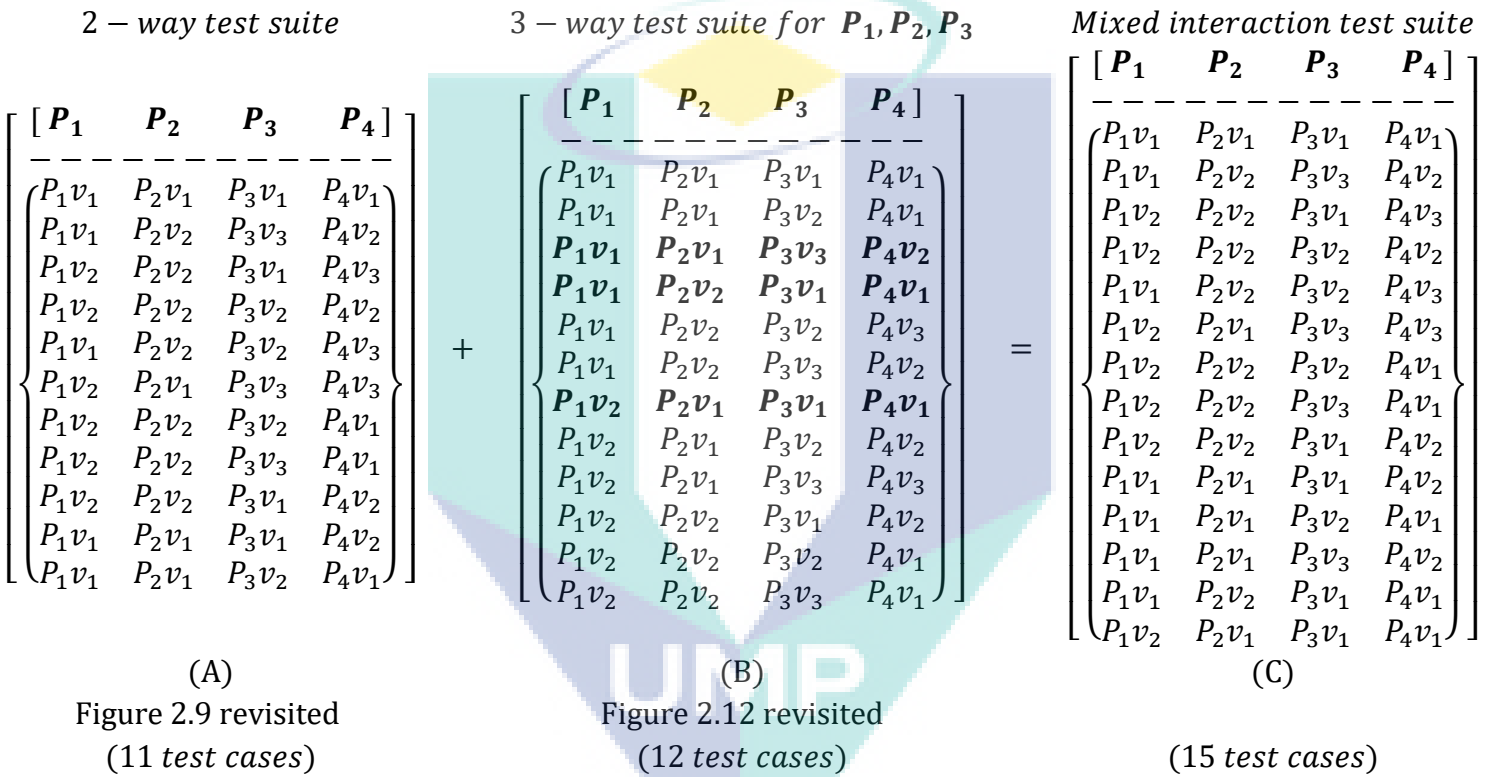


Figure 2.13 The test suite for the overall system with the mixed-strength interaction.

Referring to Figure 2.13 (C) , the total test suite has been reduced from 36 (at full interaction strength t=4) to 15 for the mixed-strength example considered (i.e. the two-way interaction with three-way sub-strength for the first three parameters (P_1, P_2, P_3)). In this case, a reduction of 58.3% has been achieved. Here, the test generated are covering all the interaction between the parameters and their values giving more focuses on the first three parameters.

In this example, each of the 2-way and 3-way pair combinations are covered at least once in the interaction tuples (at least once as evident in their occurrences). Thus, the solution given in Figure 2.9 and Figure 2.13 (C) for mixed two-way test suite and mixed-strength test suite are correct. Here, the terms "*covered*" or "*coverage*" implies the "*parameter coverage,"* All the parameters in this example are covered by the generated interaction elements. In this manner, interaction elements are the highlighted configurations in Figure 2.8 without the randomized values (i.e. 37 interaction elements are generated in the interaction tuples for the set i, ii, iii, iv, v, and vi). Thus, some of the 2-way interaction configurations are covered more than one times after the randomization process. In this case, the more interaction elements covered for the maximum available times (*occurrences*) often prevent *optimum* test suite (i.e. some of the 2-way combinations in Figure 2.8 are covered more than once). Ensuring that combinations are covered once is the key challenges in this research area, in order to get the most optimum results possible regardless of the value of interaction strength value (*t*).

As the *t* strength is further relaxed, the reduction of the test suite tends to increase. For example, the most minimum test suite for 2-way interaction strength in the example abovementioned is 11 test cases out of 36 test cases in it exhaustive case is achieved for the main-strength scenario. In the second scenario, a 15 test cases out of 36 is obtained with variable-strength. It is worth mentioning that 9 and 12 test cases can be achieved as in the both scenario (called the optimum test suite). In this example, a considerable result has been achieved. However, as highlighted in Chapter 1, pairwise (i.e. *t* = 2, as main-strength) test suite generation is considered to be insufficient to cover (or *capture*) 100% of software defects. Therefore, focusing in higher degree of *t* strength is much desired.

Most importantly, this study is using the same concept used in this example with some modification for the mixed-strength generation. In this study *t*-way test suite generation is supported up to six degrees (i.e. $max\ t_{main-strength} = 6$) with the exception of mixed-strength (or sub-strength) generation that can be represented using Equation 2.6.

$$max\ t_{sub-strength} = \begin{cases} P - 1 & , P \leq 6 \\ 6 & , 6 > P \end{cases} \quad 2.6$$

2.4 Formal Definition for *t*-way

Building from the running example in the previous section; this section formalized the terminologies that will be used throughout the thesis. These terminologies will be extensively adopted in Chapter 3.

Definition 1. A test Element (E) as a synonym for parameter (P) that consists of a set of values (i.e. $E_i \leftrightarrow \{P_i = \{v_1, v_2, \dots, v_n\}\}$). Here, the test element E is a representation of a parameter and their dependencies as a component. The test element E for parameter P_i is donated as E_i. For ease of use, as parameters are disjoint, whereas, each value v_n refers to a unique parameter. This consideration allows us to address each value v_n without referring to its parameter.

Definition 2. An Element Set (ES) that contains the test element of the software-under-test. Here, the ES for the running example is:

$$ES = \text{the value sizes for } \{E_1, E_2, E_3, E_4\}$$

Definition 3. The Element Combination (EC) (known as a t-tuple element) that contains the t-based pairing combination for the ES, here each EC$\in$ [E.sel, E.$\overline{sel}$], where $E.sel$ and $E.\overline{sel}$ imply the state of selected and not selected element from ES respectively. Here, the selected elements are ticked (see 2.14), which shows an illustration of complete 2-way tuple selection used in the running example.

$$\begin{bmatrix} ES & : & E_1 & E_2 & E_3 & E_4 \\ \text{EC 1:} & & \checkmark & \checkmark & & \\ \text{EC 2:} & & \checkmark & & \checkmark & \\ \text{EC 3:} & & \checkmark & & & \checkmark \\ \text{EC 4:} & & & \checkmark & \checkmark & \\ \text{EC 5:} & & & \checkmark & & \checkmark \\ \text{EC 6:} & & & & \checkmark & \checkmark \end{bmatrix}$$

Figure 2.14 The element t-tuple sets dominastration.

Definition 4. The Combinations *t*-Tuples Set (CTS) that is a list contains all the valid combination elements. For example, the in case of pairwise interaction, CTS = {EC 1, EC 2, EC 3, EC 4, EC 5, EC 6}, Figure 2.15 illustrates the CTS, these EC sets in CTS will use to generate all required interaction elements that cover all the test suite.

$$\begin{bmatrix} EC\ 1 & = & \{E_1, E_2\} \\ EC\ 2 & = & \{E_1, E_3\} \\ EC\ 3 & = & \{E_1, E_4\} \\ EC\ 4 & = & \{E_2, E_3\} \\ EC\ 5 & = & \{E_2, E_4\} \\ EC\ 6 & = & \{E_3, E_4\} \end{bmatrix}$$

Figure 2.15 The combinations t-tuples set (CTS) list illustration.

Definition 5. Interaction Element Tuples (IET). At based interaction element (IE) configurations set that each set covers an EC values. Each set of tuples in IET $\in [v.sel, v.\overline{sel}\ ]$, representing all the ie configurations driven by the generation of the combination dependencies on CTS list. Formally, IET is number of configurations sets that covers all CTS for the domain targeted. Each generated EC set in IET represent an "element *t*-tuple". For example, let's consider the demonstration for one EC (i.e. EC 1). Here, all the possible values for the test element E_1 and E_2 are considered. Unlike, E_3 and E_4 are considered as "don't care" values (see Figure 2.16).

$$\begin{bmatrix} \text{IE } 1 & = & \{P_1v_1, P_2v_1, x, x\} \\ \text{IE } 2 & = & \{P_1v_1, P_2v_2, x, x\} \\ \text{IE } 3 & = & \{P_1v_2, P_2v_1, x, x\} \\ \text{IE } 4 & = & \{P_1v_2, P_2v_2, x, x\} \end{bmatrix}$$

Figure 2.16 The illustration for IE set for EC 1 tuple in IET.

The example in the problem definition model section has six different valid *t*-tuple, or element combinations (EC) are indicated as ticked (✓) in Definition 3. The unselected elements are empty. In Definition 4, EC sets are formulated based on the EL list that defined the components of the software-under-test from Definition 2. Then, these EC sets is combined in a CTS set in order to generate the IE sets that construct IET Definition 5 for test suite generation process. To elaborate, each EC formulates its own representative IE set, combining all of the IE sets generated by the element *t*-tuples formulates IET. In the running example, The IET set is constructed based on six EC (or called parameter combinations) sets, each of which has its own unique interaction configurations. Each IE set in IET can be calculated using the Equation 2.7.

$$IE\ size = (v_{E_i})! \tag{2.7}$$

Next, the element E_1 and E_2 values are constructing the IE set for EC1 above-mentioned. Thus, the size of IE set for EC1 based on the Equation 2.7 is $2 \times 2 = 4$ IE in this case (i.e. $v_{E_1} = 2\ and\ v_{E_2} = 2$).

Definition 6. In order to optimize the IE sets in IET, that covers all the possible EC sets in CTS for the targeted software-under-test using BA, an objective function ($f(x)$) for optimizing IET need to be identified. This objective function can be specified as follows;

$$f(x) = \sum_{n=1}^{IET\ size} y\ covers\ IE \tag{2.8}$$

$$,where\ y = \{v_{E_1}, v_{E_2}, \dots, v_{E_i}\};\ i = 1, 2, \dots, N$$

The $f(x)$ is coverage function, $f(x)$ uses to evaluate the fitness of the covering IE by each generated solution (or test candidate (y)). y symbol implies a set of decision values (v_{E_i}) constructed based on the N, N implies an index representing a reference of the generated values of E_i in ES. IET size indicates the total number of IE in IET.

2.5 The Existing *t*-way Strategies

This section highlights the well-known *t*-way test suite generation strategies. Here, following the scope of this work, algebraic strategies (i.e. Orthogonal Arrays (OA) (Bush, 1952; Cheng, 1980; Mandl, 1985), CA, and MCA) (Cohen et al., 1994; Cohen, 2004; Williams, 2000; Williams & Probert, 1996) are not discussed further than the briefly description of CA and MCA used in the mathematical notation section (see Section 2.2), as these strategies are available for limited configurations (Cheng, 1980; Yu et al., 2008). In similar manner, our analysis also omit several of the greedy based strategies that only support low interaction strength (i.e. In Parameter Order (IPO) (Lei & Tai, 1998), the Orthogonal Array Based Testing Strategy (OATS) (Krishnan et al., 2007), G2Way (Klaib et al., 2008), IRPS (Younis et al., 2008b; Younis et al., 2010), ITTW (Younis & Zamli, 2009a), Reverse Tracking Strategy (RTS) (Younis & Zamli, 2009b), AllPairs (Bach, 2002), ReduceArray2 and ReduceArray3 (Daich, 2003), rdExpert (Copeland, 2004), SmartTest (Inc., 2014), ORA (Younis et al., 2008a), PS2Way (Khatun et al., 2011), MT2Way (Rabbi et al., 2012) and EPS2Way (Rabbi et al., 2011)). The scope of this work focuses on high interaction strength (i.e.$3< t \leq 6$).

Indeed, a number of surveys have been conducted in the last two decades. As a matter of fact, Cohen (Cohen, 2004) has surveyed the interaction test generation strategies into two main classifications, which are algebraic frameworks (i.e. OA, CA, MCA) and computational techniques including greedy algorithms frameworks, IPO, algebraic tools (i.e. TConfig (Williams et al., 2003), Combinatorial Test Services (CTS) (Hartman & Raskin, 2004a)), heuristic search (i.e. HC) and meta-heuristic search (i.e. SA, Great Deluge Algorithm, Tabu search (TS) and GA). Concerning meta-heuristic search, McMinn (McMinn, 2004) surveys the meta-heuristic search strategies including HC, SA, GA and Evolutionary Algorithms.

Well ahead, Grindal (Grindal et al., 2005) surveyed the interaction test generation based on three main groups, the first group is non-deterministic (probabilistic) where the test suite is varying in each run (i.e. Automatic Efficient Test Generator (AETG) (Cohen et al., 1994) and CATS (known as TestCover) (Sherwood, 1994, 2003)) as a heuristic non- deterministic method. Then, deterministic group with three subgroups that includes instant (i.e. OA (Mandl, 1985) and CA), iterative (i.e. Each Choice also known as 1-way (Ammann & Offutt, 1994), Partly Pair-Wise (PPW) (Burroughs et al., 1994), Base Choice

(BC) (Ammann & Offutt, 1994), default testing (Burr & Young, 1998) and Anti-random (AR)(Malaiya, 1995)), and parameter-based strategies (i.e. IPO). Lastly, the last group is the compound strategies (i.e. BCAETG), which is a compound of BC and AETG.

Most importantly, Nie (Nie & Leung, 2011) highlighted a survey that specifically highlighted mixed-strength interaction strategies. Building from Nie and Leung, Othman (Othman et al., 2013) proposed a critical survey and analysis based on the main support for *t*-way test generation strategies (i.e. supported interaction, computational implementation, automation support, strategy approach, and deployment). Othman surveys and analysis includes several strategies such as In-Parameter-Order-General (IPOG) and its variants, Modified IPOG (MIPOG), ParaOrder, ReqOrder, AETG, Test Case Generator (TCG), Jenny, Pairwise Independent Combinatorial Testing (PICT), Test Vector Generator (TVG), SA, GA and Variable Strength Particle Swarm Test Generator (VS-PSTG), Union and Greedy strategy, Generalized T-Way Test Data Generator (GTWay), Density based Strategies and TConfig. Recently, Jimena Adriana (Timaná-Peña et al., 2016) reviewed the state-of-art metaheuristic algorithms and their applications. Jimena Adriana review covered most of the well-known metaheuristic algorithms, which includes; SA, TS, GA, ACO, PSO, and HS algorithms.

In another work, Dias Neto (Dias Neto et al., 2007), Afzal (Afzal et al., 2009) and Ali (Ali et al., 2010) systematically reviewed a handful of *t*-way test suite generation. Al-Sewari and Zamli (Alsewari & Zamli, 2014), Khalsa and Labiche (Khalsa & Labiche, 2014) analyse many strategies in their orchestrated survey. While Cohen, Grindal, McMinn, Othman, Jimena Adriana, Dias Neto, Afzal, Al-Sewari and Zamli Khalsa and Labiche have surveyed and analysed the state-of-the-art available *t*-way test suite generation strategies at the period of their work, their work has not considered recent developments especially on the application of newer meta-heuristic based strategies. Thus, this work extends existing reviews and surveys to include and highlight handful of a newly developed of *t*-way strategies.

The key aspect discussed in this section is to observe the strength and limitation of the existing strategies in terms of the interaction strength degree supported and the method used to generate test suite in addition to the support of mixed-strength interaction. Here, the strategies are classified based on the consistency of the output (test suite generated) to two main groups; deterministic (i.e. the same test suite in each run) and

non-deterministic (or *probabilistic* whereby different test suite are generated in each run owing to the randomness behaviour of test selection) *t*-way strategies (see Figure 2.17).

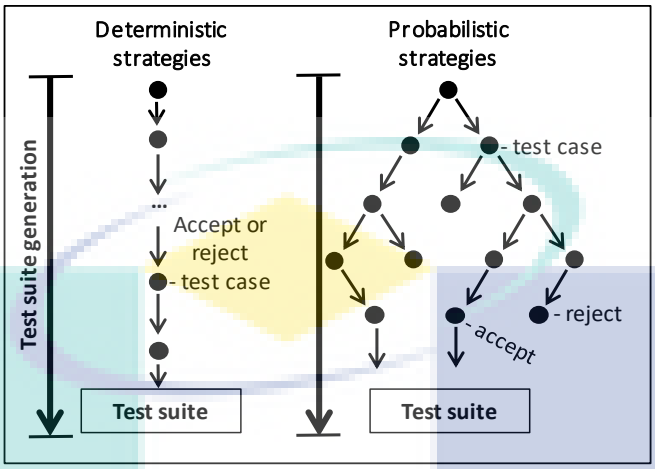


Figure 2.17 The illustration of the deterministic and probabilistic process.

The following sub-sections fulfil the main aspect in this section by highlighting existing *t*-way test suite generation strategies based the output of the test suite into two main categories, which are the deterministic and probabilistic *t*-way test suite strategies. Furthermore, following the scope of this work (to design, implement and evaluate a *t*-way strategy based in swarm meta-heuristic algorithm, which is BA) the second category is divided into seven groups as well.

2.5.1 Deterministic *t*-way Test Suite Generation Strategies

In this sub-section, the first category, the deterministic strategies, of the *t*-way test generation strategies is discussed. All the *t*-way strategies in this category considered to be greedy strategies except TConfig (Williams et al., 2003) which can also be considered as algebraic and greedy strategies (i.e. TConfig uses several methods of test suite generation). These strategies considered to be flexible for test generation, however, the optimum test suite is mostly not achievable for most of the configurations in this category.

2.5.1.1 Greedy Strategies

To start with, TConfig (Williams et al., 2003) uses two different generation algorithms to construct its test suite; the first algorithm is recursive block method. Recursive block method generates test suite using algebraic approach, which means mathematical formula involves to generate *t*-way test suite, it contains an algorithm to

generate orthogonal arrays (OA) to initial blocks for the large covering array (Sherwood et al., 2005). in this method, the *t*-way test suite is generated by constructing a covering arrays from orthogonal arrays based on mathematical specifications (Williams, 2000; Williams & Probert, 2002). Unlike Recursive block, the second methods implore a modified IPOG strategy (Lei et al., 2007) to generate a high *t*-way strength interaction.

As IPOG (Lei et al., 2007) is essential for TConfig, IPOG is a generalization the pairwise approach used in IPO (Lei & Tai, 1998). A number of *t*-way strategies have been developed based on the concepts of IPOG (i.e. IPOG-D (Yu et al., 2008, 2009), IPOG-F (Forbes et al., 2008) ,MIPOG (Younis & Zamli, 2010b; Younis & Zamli, 2011), MC-MIPOG (Younis & Zamli, 2010a) and ParaOrder (Wang et al., 2008)), which called IPOG family use a vertical and horizontal extension similar to IPO. The test suite generation process starts with building the pairwise tests for the first parameter then extend to the other parameters and so on until all the parameters are covered. When the horizontal extension is not possible, the uncovered interactions are covered using a vertical extension. If needed. IPOG family supports high interaction strength (i.e. $t \leq 6$) except MIPOG that supports interaction strength up to 12. Unlike MIPOG, ParaOrder (Wang et al., 2008) only provides the support up to 3 interaction strength (i.e. $t \leq 3$). On a positive note, ParaOrder allows prioritization of *t*-way interaction for its horizontal extension. The extended parameter for ParaOrder strategy is decided based on a number of values (i.e. parameter with the higher number of values will be extended first). However, only IPOG, IPOG-F, and ParaOrder supports mixed-strength interaction. Unlike IPOG, TConfig does not support for mixed-strength interaction.

Jenny (Pallas, 2003) adopts a greedy algorithm to generate *t*-way test suite. Jenny starts with 1-way generation, then, proceeds with two-way generation up to n^{th}-way specified by the user to cover all the test interactions. Jenny does not support mixed-strength interaction. Complementing Jenny, IBM Intelligent Test Case Handler (ITCH) (Hartman et al., 2005), was developed by IBM Haifa and Watson Research Laboratories. ITCH considered being an improvement of Combinatorial Test Services (Hartman & Raskin, 2004a) which only support two-way interaction. ITCH gives the user the ability to control the test suite size. ITCH uses an exhaustive search algorithm to generate *t*-way test suite up to 4 interaction strength. There is no evidence in the literature as to whether or not ITCH supports mixed-strength interaction.

Another strategy is GTWay (Klaib, 2009; Klaib et al., 2008; Klaib et al., 2015). GTWay employed three algorithms to generate *t*-way test suite. The first algorithm is a parser algorithm that constructs the SUT parameters and values as symbolic representation pairs to be used for *t*-way configurations generation. Then, the *t*-way pair generation algorithm generates the *t*-way interactions based on the configuration pairs from the parser algorithm. Finally, the backtracking algorithm generates the *t*-way test suite by iteratively combining the parameters values in the interaction tuples generated by the *t*-way pair generation algorithm. GTWay does not support variable-interaction. GTWay meant to address a high interaction strength (i.e. $t < 12$).

2.5.2 Probabilistic *t*-way Test Suite Generation Strategies

In this section, the second category, the probabilistic *t*-way test generation strategies are discussed. Given the effectiveness of probabilistic-based strategies than its deterministic counterparts in term of test suite reduction, it is not surprising that many researchers are focusing more on probabilistic approach. As *t*-way test suite generation can be viewed as a combinatorial optimization problem, many strategies for *t*-way test suite generation has emerged based on a meta-heuristic algorithm (i.e. GA, CS, HS, PSO, SA, and HC) as the backbone search engine. As the name suggests, meta-heuristic algorithm is dedicated algorithm to find the most optimal result for the targeted domain, which in this case test suite generation.

In the following sub-sections, the probabilistic *t*-way strategies are grouped and briefly elaborated based on the main generation technique used for minimizing the test suite. These strategies can be divided into seven groups based on the method of optimization process as follows; Greedy, Evolutionary, Simulated Annealing, Harmonic, Stochastic, Tabu, and Swarm strategies. Here, some of the well-known pairwise strategies will be covered as this category follows the scope of this work, as BA is categorized as swarm based optimization algorithm (Yang, 2014; Yang & Gandomi, 2012). Thus, some of the swarm optimization based pairwise strategies are also included.

2.5.2.1 Greedy Strategies

This section describes the probabilistic greedy strategies as an effort to complement the review of greedy strategies for the deterministic category.

AETG (Cohen et al., 1997; Cohen et al., 1994) can be considered as a pioneer strategy for *t*-way test reduction. AETG, developed by Cohen (Burr & Young, 1998; Cohen et al., 1997; Cohen et al., 1994; Dalal et al., 1999; Dalal et al., 1998; Ellims et al., 2008), uses a greedy algorithm. AETG considered to be the first *t*-way test generation that is commercially available (Cohen, 2011). In this strategy, an empty test suite is defined. Then, ATEG starts its generation process. In each iteration, ATEG generates a set of test cases. Here, the best test case that covers the maximum number of the uncovered interaction elements (*t*-way interaction tuples) is selected and added to the test suite. AETG does not support mixed-strength interaction, and there is no published evidence to high interaction support (i.e. *t* equals 4, 5, or 6). Concerning implementation, AETG provides a reusable software component. For this reason, several variants of AETG have been developed including mAETG (Cohen, 2004) and mAETG_SAT (Cohen et al., 2007b). mAETG is the AETG modification strategy that supports mixed-strength interaction. In addition to mAETG, Myra (Cohen, 2004) modified Test Case Generator (TCG) (Tung & Aldiwan, 2000) called mTCG. TCG considered being an open source variant of AETG. Generally, both TCG and mTCG improve the performance of the original AETG.

Similar to AETG, Test Vector Generator (TVG) (Arshem, 2003) implements a public domain strategy supporting *t*-way test suite generation (as TVG claimed to be ATEG variant). TVG exploits three algorithms namely, T-reduced algorithm, Plus-one algorithm, and Random sets algorithm for *t*-way generation. Although useful, not much information can be implied as the details implementation has not been made available in the literature. TVG generates test suite with a high value of interaction strength (i.e. $t \leq 6$). TVG also addresses mixed-strength interaction.

Another well-known strategy, PICT (Czerwonka, 2006; Othman et al., 2013) is a public *t*-way strategy developed by Microsoft. This strategy is able to generate a *t*-way test suite that supports mixed-strength interaction. By generating all the interaction tuples then matching the interaction combinations with their test cases randomly, PICT often generates non-optimal results.

Finally, Classification-Tree Editor eXtended Logics CTE-XL (Lehmann & Wegener, 2000) is a *t*-way test suite generation strategy based on Classification-Tree Method (CTM). Here, CTE-XL uses the CTM approach to classify the test data into classes based on the test specifications. The test cases are generated by classes from different classifications. CTE-XL has been enhanced through the time (Yu et al., 2003). However, CTE-XL supports low interaction strength (i.e. $t \leq 3$). There is no support for mixed-strength interaction.

2.5.2.2 Evolutionary Strategies

Evolutionary strategies, as the name suggest, is based on the evolutionary metaheuristic algorithms such as GA, genetic programing and evolutionary programing (Afzal et al., 2009; Bansal et al., 2015; Bryce & Colbourn, 2007; Shiba et al., 2004; Sthamer, 1995). These strategies are derived from the survival behaviour of the fittest individuals (natural selection process). Usually, these strategies use selection, crossover, mutation and replacement as their test suite generation process. They start with a set of test cases candidates that processed to generate a subset of the test cases (test suite) with the highest covering value. Generally, evolutionary strategies start with randomly generated test candidates that reflect chromosomes construction. Then, the chromosomes are processed. In each cycle, a crossover and mutation processes are undergoing to meet the best fitness of the predefined function. The best fitness chromosomes are considered to be the optimum test suite.

Several evolutionary strategies have been proposed including PWiseGen (Flores & Cheon, 2011), G-PWiseGen (Sabharwal et al., 2017), PWiseGen-GM (Sabharwal et al., 2015), Pairwise test set generator using genetic algorithm (PTSG-GA)(Sabharwal et al., 2016), PWiseGen-VSCA (Bansal et al., 2015), GA-Huang (Huang et al., 2010), Genetic Algorithm for Pairwise Test Sets (GAPTS) (McCaffrey, 2009a, 2010), QICT (McCaffrey, 2009c), Weight-Based GA (WBGA) (Wang et al., 2013), Nondominated

Sorting Genetic Algorithm II (NSGA-II) (Deb et al., 2002), A cellular genetic algorithm for multi-objective optimization (MOCell) (Nebro et al., 2009), A Parallel Genetic Algorithm based on Spark (PGAS) (Qi et al., 2016), and Evolutionary Genetic Algorithm (EGA) (Lopez-Herrejon et al., 2016). Mostly, the existing evolutionary strategies are capable of generating the low value of uniform interaction strength (i.e. $t \leq 3$) with the exception of G-PWiseGen which higher interaction strength (i.e. $t \leq 4$). Here, PWiseGen-VSCA is the only GA-based strategy that covers mixed-strength interaction although supporting low interaction strength (i.e. $t \leq 3$).

2.5.2.3 Simulated Annealing Strategies

Unlike evolutionary strategies, Simulated Annealing (SA) strategies (Bryce & Colbourn, 2007; Cohen et al., 2007b; Cohen et al., 2008b; Stardom, 2001) (Chen & Chien, 2011; Cohen et al., 2007b) adopt SA as the base of their test suite generation. SA strategies use a stochastic optimization method in general. Specifically, SA strategies are based on the metals annealing process, which use to obtain materials that are more resilient and possess better qualities for industry applications. Basically, this process start with melting the material at a specific temperature to reach its liquid state, in order to increase this material atoms mobility within the structure. Then, a cooling process undergoes until the temperature reaches a stopping condition. By each cooling process, the atoms lose their mobility to achieve thermal equilibrium at the end of the process. Thus, a highly stable material structure is produced. SA strategies rely on generating highly random test candidates, which are accepted based on probability-based transformations equations. Here, in each iteration, test candidates are checked to ensure the test candidate covers the highest number of *t*-tuples when it reaches the highest possible, a cooling schedule is applied. Then, the transformation is accepted. The candidate that covers largest number of *t*-tuples is selected and added as a test case. Finally, a test suite is generated at the end of the iterations.

Several SA strategies have been developed including SA-Mayer (Mayer implementation of SA) SA_SAT (Cohen et al., 2007b; Cohen et al., 2008b), Augmented Simulated Annealing (ASA) (Cohen et al., 2003b), These strategies often adopt binary search algorithm to generate the test suite. Recent work includes SA-H (George, 2012) and the Simulated Annealing algorithm for constrained Combinatorial interaction testing (CASA) (Garvin et al., 2009; Garvin et al., 2011) and the improved CASA (tCA)

(Haslinger et al., 2013). These strategies support low value of interaction strength (i.e. $t \leq 3$). Another strategy, SA-Bryce (Bryce implementation of SA) (Bryce & Colbourn, 2007) and EDIST-SA (Rahman et al., 2015) demonstrates a 4-way interaction strength support. The Improved SA (ISA) (Torres-Jimenez & Rodriguez-Tello, 2012) employ a binary alphabet algorithm to cover a high interaction strength (i.e. $t \leq 6$). Recently, a hybrid Simulated Annealing Variable Neighbourhood Search (SAVNS) (Rodriguez-Cristerna et al., 2015) that is an improved version of SA-VNS (Rodriguez-Cristerna & Torres-Jimenez, 2012), employs SA with the variable neighbourhood search function to construct mixed-strength interaction up to $t = 5$. Similar to SAVNS, SA-Mayer, SA_SAT, and ASA are supporting mixed-strength interaction as well.

2.5.2.4 Harmony based Strategies

Harmony based strategies (Alsewari & Zamli, 2012a; Bao et al., 2015; LI et al., 2013; Xiang et al., 2015) are based on the musical improvisation process employed in the harmony search algorithm. Harmonic strategies use global and local search to construct test suite. Here, Harmony Search Strategy (HSS) (Alsewari & Zamli, 2012a) considered being the state-of-are of harmonic strategies. HSS is an extended version of the Pairwise Harmony Search Strategy (PHSS) (Alsewari & Zamli, 2012b). Harmonic strategies adopt two probability values (i.e. the considering rate and pitch adjustment rate). Here, global search is iteratively performed by randomizing values in the Harmony memory whereby the local best value can be selected given a considering rate probability. Here, the local best value can be considered for improvements for further improvements in the local search (i.e. with pitch adjustment probability). Upon completing each iteration, the best value will be added to the final test suite until all pairwise interactions are covered. PHSS and the Harmony Search-Pairwise Test Suite Generator Tool (HS-PTSGT) (Xiang et al., 2015) are supporting only pairwise interaction strength, in contrast, Improved HS (IHS) (Bao et al., 2015) supports up to 6. on the other hand, HSS and Harmony Search Test Suite Generator (HSTSG) (LI et al., 2013) are supporting mixed-strength interaction and high interaction strength (i.e. $t \leq 7$ and $t \leq 15$, respectively) as well.

2.5.2.5 Stochastic Hill Climbing Strategies

Stochastic (*Hill climbing*) strategies (Alsewari et al., 2014, 2015; Bryce & Colbourn, 2007; Cohen et al., 2003a; Nasser et al., 2014; Stardom, 2001; Zamli et al., 2015) is based on the hill climbing algorithm. Essentially, hill climbing constructs a test suite using a very simple method based on a series of transformations. Generally, hill climbing based strategies start with a random test case that evaluated based on the number of interaction sets that is not covered. Then, in the next iteration, a transformation for the current test case is processed where one of it parameters value is randomly regenerated based on the values in that position. This process continues until all the interaction sets are covered. Several strategies have been developed using based on the hill climbing including HC-Bryce (Bryce & Colbourn, 2007) that covers considerable interaction strength (i.e. $t \leq 4$) and the Late Acceptance Hill Climbing (LAHC) strategy (Alsewari et al., 2014, 2015; Nasser et al., 2014; Zamli et al., 2015) that employ a memory that is randomly initial as a population of test cases. LAHC support mixed-strength interaction as well as a high interaction strength (i.e. $t < 6$).

2.5.2.6 Tabu Strategies

Tabu strategies (Bryce & Colbourn, 2007; Zamli et al., 2016; Zekaoui, 2006) are based on Tabu Search (TS). TS can avoid help to reduce the possibility of falling local minimum, Specifically, TS uses a temporary memory (tabu list) to avoid returning to past solutions. Several strategies have evolved based on TS including TS-Bryce (Bryce & Colbourn, 2007), TSA (or MiTS) (Gonzalez-Hernandez, 2015; Gonzalez-Hernandez et al., 2010), PAT (or POT) (Zekaoui, 2006) and High Level Hyper-Heuristic (HHH) (Zamli et al., 2016). Unlike TS-Bryce and PAT which support only for small interaction strength ($t \leq 4$), HHH and MiTS address the full support until $t \leq 6$. Unlike other strategies, HHH adopts TS to leverage on the strength of four other algorithms for better test suite generation. MiTS is the only strategy that provides the support for mixed-strength interaction.

2.5.2.7 Swarm Strategies

Swarm strategies (Ahmed et al., 2014; Ahmed & Zamli, 2010a; Ahmed et al., 2012b; Chen et al., 2010) mimic the behaviour or movements of organism swarms in nature (i.e. ants, bees, shoals of fish or flocks of birds). Several strategies have been implemented based on many optimization algorithms (e.g., Particle Swarm Optimization (PSO), Simplified Swarm Optimization (SSO), Ant Colony System (ACS), Ant Colony Optimization (ACO), Artificial Bee Colony (ABC), Simulated Bee Colony (SBC) and Cuckoo Search (CS)). Notably, PSO appears to be the most popular swarm based optimization algorithm adopted for *t*-way test generation.

Many strategies employ PSO for test suite generation (Ahmed & Zamli, 2010a; Ahmed et al., 2012b; Chen et al., 2010). These strategies took an advantage of the local and global search performed by PSO to construct optimal test suite including PSO-Chen (Chen et al., 2010), TDGen_PSO (Mao et al., 2012), Discrete Particle Swarm Optimization (DPSO) (Jia-Ze & Shu-Yan, 2012) and Pairwise Particle Swarm Test Generator (PPSTG) (Ahmed & Zamli, 2011a) are strategies that are proposed for pairwise test suite generation. PPSTG is then improved to a *t*-way strategy (i.e. Particle Swarm Test Generator (PSTG) (Ahmed & Zamli, 2010a; Ahmed et al., 2012b) and Variable Strength Particle Swarm Test Generator (VS-PSTG) (Ahmed & Zamli, 2011b)). Recently, a new implementation of PSO that uses fuzzy logic to tune the its heuristic parameters called FSAPSO (Mahmoud & Ahmed, 2015) has been proposed. In addition, SITG (Rabbi et al., 2015) has introduced a new *t*-way strategy based on PSO. SITG, PSTG, VS-PSTG and FSAPSO support ideal interaction strength (i.e. $t \leq 6$). On other note, VS-PSTG supports mixed-strength interaction as it implemented for this reason.

Similarly, the SSO adopts a simplified version of PSO (also known as Many Optimization Liaisons (MOL)) (Ahmed et al., 2014). SSO strategy merely supports low interaction (i.e. $t \leq 3$). Based on this idea, many swarm strategies have emerged such as bees-based strategies which include, Bees Algorithm (BA*) strategy (Zabil & Zamli, 2013a; Zabil & Zamli, 2013b; Zabil et al., 2012), Artificial Bee Colony (ABC) (Mala & Mohan, 2009), Artificial Bee Colony-Covering Array Generator (ABC- CAG) (Bansal et al., 2016) and Simulated Bee Colony (SBC) (McCaffrey, 2009b). These strategies employ a bee family algorithm, which considered as a swarm algorithm. Generally, these algorithms based on the foraging (search for food) behaviour of bee colonies. Here, the

test cases represented as bees. The test suite generation starts with the random population for scout bees that search for a food source. The paths that bees visited are evaluated. The bees that cover the highest number of *t*-tuples by each iteration are selected to be a part of the test suite. BA shows high strength support (i.e. $t \leq 10$). Unlike ABC, ABC- CAG and SBC, that only support low interaction strength (i.e. $t = 2$).

Another type of swarm strategies are ants-based strategies (Chen & Chien, 2011; Chen & Zhang, 2009; Shiba et al., 2004), these strategies based on the forging behaviour of ant colonies. They employ the Ant Colony System (ACS) (Dorigo et al., 1989; Drigo et al., 1996) algorithm in general and its variant (i.e. Ant Colony Optimization (ACO)(Dorigo et al., 2006)) later on. Ants-based strategies simulate the communication that ants use to determine the shortest route between their colony location (starting location) and food sources. The test cases in these strategies are represented as a route from a start point to an end point (target). In other words, each test cases and its interaction coverage are combined into one element, called path. The quantity of pheromones left in each path visited by ants reflects the quality of the solution. The path with the largest quantity of pheromones and highest probability is chosen as an optimal solution (best test case).

The most well-known ants-based strategies include the Variable Strength Interaction Test suites (ACS-VSITs) strategy (Xiang et al., 2009), ACS-VSITs strategy adopts ACS, as a variant of ACO. Another strategy is Prioritized pairwise Interaction Test Suite (PITS) strategy (Chen et al., 2009) that adopts ACO as a general base algorithm. Actually, PITS generates test suite based on four variants of ACO, which are; Ant System, Ant System with Elitist, Ant Colony System, and Max-Min Ant System. Shiba (Shiba et al., 2004) also has proposed a strategy called Ant Colony Algorithm (ACA). ACA-Shiba constructs test suite using a combination of ACO and AETG. Based on ACO, ACA-Chen (Chen et al., 2009) shows a better competitive results. However, ACA-Chen performance could not be generalized as it only supports pairwise interaction. Here, ACA-Shiba, PITS and ACS-VSITs support the low value of interaction strength (i.e. $t \leq 3$), only VSITs supports mixed-strength interaction.

The last type of swarm strategies is the cuckoo-based strategies (Ahmed et al., 2015; Nasser et al., 2015) based on the Cuckoo Search (CS) Algorithm. cuckoo-based strategies (i.e. CS strategy (Ahmed et al., 2015) and Pairwise-CS (Nasser et al., 2015)) start by initializing a population of random test candidates (nests). Each nest consists of the random parameter values that represent test case candidates. Then, test cases are evaluated using a fitness function (i.e. based on the number of test case covered in the interaction sets (*t*-tuples)). Here, a test case candidate in each nest with the highest fitness value considered as a test case. The global search process in cuckoo-based strategies uses a Lévy flight transformation between the nests to determine the nest with the highest test case coverage. This process continues until all the *t*-tuples are covered. Pairwise-CS only covers pairwise interaction strength; in contrast, CS strategy supports high interaction strength (i.e. $t \leq 6$). However, cuckoo-based strategies have not covered the support for a mixed-strength interaction.

2.5.3 The Observation of the Highlighted *t*-way Strategies

This sub-section illustrates an analysis of the features the well-known *t*-way strategies (see Figure 2.18). Figure 2.18 illustrates an analysis of the features that are commonly shared by each strategy and those that are not. The existing *t*-way strategies are firstly divided into two major categories; deterministic or probabilistic strategies. After that, the probabilistic strategies are divided into seven groups based in the method employed for test suite constriction. Figure 2.18 divides the interaction strength support into three groups; $t \leq 3$, $3 < t \leq 6$ and $t > 6$ and highlights them with yellow, green and red, respectively. The mixed-strength supported strategies are highlighted as well with grey colour. Additionally, the swarm strategies are also divided into groups based on the swarm algorithm employed. A closer look to the Figure 2.18 shows that Bat-inspired algorithm (BA) has not been adopted in this field.

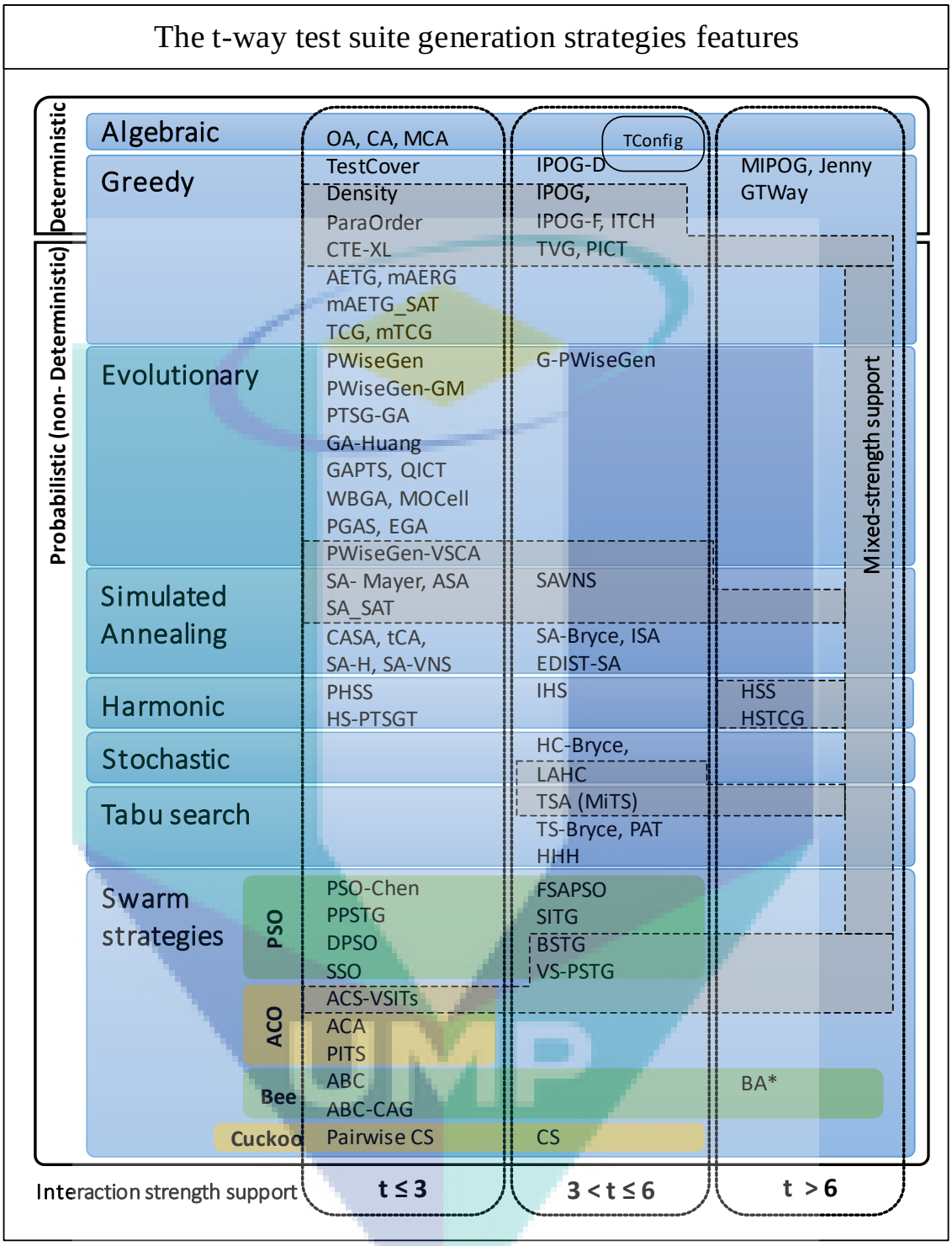


Figure 2.18 Features of the existed *t*-way test suite generation strategies.

2.5.4 The Justification of the Adoption of BA

As reported throughout the *t*-way testing literature in section 2.5, *t*-way interaction test generation has indeed achieved considerable progress. However, the investigation for new test suite generation strategies is deemed necessary (Yu et al., 2008) to achieve more effective *t*-way test suite. Table 2.3 summarizes the description of existing t-way strategies.

Table 2.3 The analysis of existing t-way strategies.

Supported ✓ / Not supported ×		Randomness		Interaction strength			Mixed-strength support	Enhanced search algorithm
Strategy family	**Existed t-way strategies**	**Deterministic**	**Non-Deterministic**	**t≤3**	**3 < t < 6**	**6 ≤ t**		
Greedy	AETG	✓	×	✓	×	×	×	×
	mAETG	✓	×	✓	×	×	×	×
	mAETG_SAT	✓	×	✓	×	×	×	×
	TVG	✓	×	✓	✓	✓	✓	×
	CTE-XL	✓	×	✓	×	×	✓	×
	PICT	✓	×	✓	✓	✓	✓	×
	MIPOG	✓	×	✓	✓	✓	×	×
	GTWay	×	✓	✓	✓	✓	×	×
	Jenny	×	✓	✓	✓	✓	×	×
	ITCH	×	✓	✓	✓	×	×	×
	TestCover	×	✓	✓	×	×	×	×
	Density	×	✓	✓	×	×	✓	×
	IPOG	×	✓	✓	✓	✓	✓	×
	IPOG-D	×	✓	✓	✓	✓	×	×
	IPOG-F	×	✓	✓	✓	✓	✓	×
	ParaOrder	×	✓	✓	×	×	✓	×
Algebraic	OA	×	✓	✓	×	×	×	×
	CA	×	✓	✓	×	×	×	×
	MCA	×	✓	✓	×	×	×	×
	TConfig	×	✓	✓	✓	✓	×	×
Evolutionary	PWiseGen	×	✓	✓	×	×	×	×
	G-PWiseGen	×	✓	✓	✓	×	×	×
	PWiseGen-GM	×	✓	✓	×	×	×	×
	PTSG-GA	×	✓	✓	×	×	×	×
	PWiseGen-VSCA	×	✓	✓	×	×	✓	×
	GA-Huang	×	✓	✓	×	×	×	×
	GAPTS	×	✓	✓	×	×	×	×
	QICT	×	✓	✓	×	×	×	×
	WBGA	×	✓	✓	×	×	×	×
	MOCell	×	✓	✓	×	×	×	×
	PGAS	×	✓	✓	×	×	×	×
	EGA	×	✓	✓	×	×	×	×

Supported	✓
Not supported	×

Strategy family	Existed t-way strategies	Randomness		Interaction strength			Mixed-strength support	Enhanced search algorithm
		Deterministic	Non-Deterministic	t ≤ 3	3 < t < 6	6 ≤ t		
Simulated Annealing	SA- Mayer	×	✓	✓	×	×	✓	×
	SA_SAT	×	✓	✓	×	×	✓	×
	ASA	×	✓	✓	×	×	✓	×
	SA-Bryce	×	✓	✓	✓	×	×	×
	CASA	×	✓	✓	×	×	✓	×
	SA-VNS	×	✓	✓	×	×	✓	×
	ISA	×	✓	✓	✓	✓	×	×
	SA-H	×	✓	✓	×	×	×	×
Harmonic	HSS	×	✓	✓	✓	✓	✓	×
	PHSS	×	✓	✓	×	×	×	×
	HSTCG	×	✓	✓	✓	✓	✓	×
	HIS	×	✓	✓	✓	✓	×	×
	HS-PTSGT	×	✓	✓	×	×	×	×
Stochastic	HC-Bryce	×	✓	✓	✓	×	×	×
	LAHC	×	✓	✓	✓	×	✓	×
Tabu Search	TS-Bryce	×	✓	✓	✓	×	×	×
	TSA (MiTS)	×	✓	✓	✓	✓	✓	×
	PAT	×	✓	✓	✓	×	×	×
	HHH	×	✓	✓	✓	✓	×	×
Swarm Strategies	PSTG (PSO)	×	✓	✓	✓	✓	✓	×
	VS- PSTG (PSO)	×	✓	✓	✓	✓	✓	×
	PSO-Chen (PSO)	×	✓	✓	×	×	×	×
	PPSTG (PSO)	×	✓	✓	×	×	×	×
	TDGen_PSO (PSO)	×	✓	✓	×	×	×	×
	DPSO (PSO)	×	✓	✓	×	×	×	×
	FSAPSO (PSO)	×	✓	✓	✓	✓	×	×
	SITG (PSO)	×	✓	✓	✓	✓	×	×
	SSO (PSO)	×	✓	✓	×	×	×	×
	ACA (ACO)	×	✓	✓	×	×	×	×
	ACS-VSITs (ACO)	×	✓	✓	×	×	✓	×
	PITS (ACO)	×	✓	✓	×	×	×	×
	BA* (Bee)	×	✓	✓	✓	✓	×	×
	ABC (Bee)	×	✓	✓	×	×	×	×
	ABC- CAG (Bee)	×	✓	✓	×	×	×	×
	SBC (Bee)	×	✓	✓	×	×	×	×
	CS-Ahmad (CS)	×	✓	✓	✓	✓	×	×
	Pairwise CS (CS)	×	✓	✓	×	×	×	×

From Table 2.3, it can be seen that the existing *t*-way strategies based on meta-heuristic algorithms have not covered the implementation of BA as *t*-way and mixed-strength strategy. BA superiority has been confirmed against several state-of-art meta-heuristic algorithms. For instance, Senthilnath (Senthilnath et al., 2016) has verified the

high performance of BA compared to GA, PSO and Bat-K-Means (BKM) for solving crop type classification problems for satellite image. Gherbi (Gherbi et al., 2014) confirmed that BA produces significantly better results than most popular optimization algorithms, including the GA, SA and PSO. Moreover, BA is easy to implement, and its parameters are highly adjustable to fit many engineering solutions (Taha et al., 2013).

In other works, Khan and Sahai (Khan & Sahai, 2012) report that the BA outperformed PSO and the GA for training Artificial Neural Networks (ANNs) within the e-learning context. Yang (Yang, 2010) also demonstrated that the BA achieves the best performance in contrast to PSO and the GA in terms of standard benchmark functions. In fact, Yang proves that PSO and HS could be considered as the generalization of the BA. Moreover, Sureja (Sureja, 2012) demonstrates that BA yields better solutions in comparison with the PSO, GA, and firefly algorithm. Similarly, a comparative study (Hegazy et al., 2015) on the BA, PSO, ANNs, Artificial Bee Colony (ABC), modified cuckoo search, support vector machine, and Flower Pollination Algorithm (FPA) confirms that BA is a superior meta-heuristic algorithm according to the results presented.

Swarm strategies have not covered mixed-strength up to t = 6 with the exception of PSO, although researchers proves that BA performs better than PSO (Gherbi et al., 2014; Khan & Sahai, 2012; Taha et al., 2013). Hence, adopting a superior meta-heuristic algorithm such as the BA could be effective to improve the state-of-the-arts. Generally, exploring the advantages of the new meta-heuristic algorithm could be advantageous to highlight its strengths and limitations for *t*-way test generation. For these reasons, the adoption of BA for *t*-way test suite generation is deemed a useful endeavour. Additionally, the adoption of Hamming distance classifier is deemed necessary in order to improve the exploration of BA (Gonzalez-Hernandez, 2015).

2.6 Summary

In this chapter, the well-known test case design strategies are reviewed. Then, the mathematical notations for *t*-way test suite generation have been elaborated based on the covering array. Then after that, a simple running example has been adopted to illustrate the problem of *t*-way test suite generation with mixed-strength demonstration as well as the validation of the interaction coverage. Next, a survey of the state-of-art of existing *t*-way strategies have been presented. Finally, an overview of the Bat-inspired algorithm along with the justification for implemented a new strategy called Bat-inspired Testing Strategy (BTS) is provided. BTS is aimed to address the *t*-way test suite generation for the up to the ideal interaction strength (t up to 6) and its special case of mixed-strength test suite generation.

Building on the presented contents in this chapter, the next chapter discusses the design of BTS. Additionally, the chapter will also outline how BA is being used as the backbone for BTS.

CHAPTER 3

RESEARCH METHODOLOGY

In the previous chapter, the concept of the *t*-way, its notations, survey, and analysis of the existing strategies were introduced. Finally, the adoption of BA as a basis to a *t*-way strategy proposed in this work has been justified.

This chapter presents an overview of the design methods of the current study. This chapter specifically describes the design and implementation of the BTS strategy, including its three phases, which are input analysis, interaction generation, and test suite generation. Additionally, the relevant BTS variables were calibrated to achieve the best possible results. At the end of this chapter, the details of the BTS prototype is also presented.

3.1 The Original BA Algorithm

For the BTS strategy, BA was used as the backbone algorithm to achieve the most optimal test suite sizes. For a better understanding of the BTS principle, it is necessary to discuss the process of BA and their flow as earlier discussed by Yang (2010).

The BA algorithm was developed based on the observation of the hunting behaviour of micro-bats in nature. To mimic the behavior of micro-bats in the simplest way, some approximations were necessary. Yang (2010) idealized three assumptions as follows:

i. For sensing the distance ahead of their flight paths, micro-bats use echolocation, and they have their unique instinct to distinguish a target of food/prey from the background barriers.

ii. During hunting, bats may travel in a random manner at a velocity v_i at position x_i with a combination of sensing frequency Q_{min}, varying wavelength λ and loudness A_0 to hunt for prey. The frequency (or wavelength) of their emitted pulses can be automatically adjusted, and the pulse emission rate $r \in [0, 1]$ can also be fine-tuned automatically, giving the current proximity of their target.

iii. The loudness of echoes from a micro-bat can be assumed to decay over time from a large and positive amplitude A_0 to a small constant value A_{min} (Yang, 2010).

These assumptions allow the impementation of BA (Algorithm 1) as shown in the Figure 3.1.

Algorithm 1: Bat-inspired Algorithm (BA)	
Input: objective function $f(x_i), x_i = (x_{i_1}, \dots, x_{i_D})^T$.	
Output: Best fitness x_*.	
1:	Define $n, T_{max}, Q_i \in [Q_{min}\ ,\ Q_{max}]$;
2:	Randomly Initialize x_i, $velocity_i$, $Q_i$for $i = 1, 2, \dots\ ,\ n$;
3:	Initialize pulse rates r_i and the loudness A_i;
4:	**while** $(ts < \mathrm{T}_{max})$ **do**
5:	**for** each bat $\mathrm{n_i}$ **do**
6:	Generate new solutions by adjusting frequency, update velocity and location using motion equations (4-2 to 4-4);
7:	**if** $(rand(0,1) > r_i)$ **then**
8:	Select the best solution in the current population;
9:	Generate a local solution around the best solution;
10:	**End**
11:	Generate a new solution by flying randomly;
12:	**if** $(rand(0,1) < A_i$ and $f(x_i) < f(x))$ **then**
13:	Accept the new solutions;
14:	Increase r_i and reduce A_i based on the tolerance;
15:	**End**
16:	Rank the bats and find the current best;
17:	**End**
18:	**End**
19:	process results and visualization;

Figure 3.1 The BA pseudo code.

Source: Yang (2010).

The BA algorithm in Figure 3.1 shows the pseudo code of BA based on the implementation for global optimization problems (Yang, 2010). The BA algorithm starts by defining its variable settings and objective functions (the problem that needed to be solved). It then, initializes its population variables. Here, for each bat (n) in the

population, the BA randomly initializes a set of variables as follows; initial location (solution) (x_i), initial velocity ($velocity_i$) and initial frequency (Q_i), at each time step (ts). The cycle of iterations in the BA is referred to as the number of generations (T_{max}). The maximum iteration refers to the maximum cycle of searches in the BA, which can be calculated based on the value of the multiplication of the number of generations with the number of bats' population (Equation 3.1).

$$Max_{cycle\ of\ iteration} = T_{max} \times n \tag{3.1}$$

The next step is the iteration process for the maximum cycle. This second step is called the movement of the virtual bats n_i. These virtual bats are derived from the bat motion equations (Equations 3.2 to 3.4). In the motion equations, the location (x_i) and velocity ($velocity_i$) of each virtual bat are updated based on the updated frequency (Q_i) by each cycle of iterations. Here, the pace and range of the virtual bats' movement are basically controlled by Q_i, which is similar to the movement of the swarming particles as follows:

$$Q_i = Q_{min} + (Q_{max} + Q_{min})\ rnd \tag{3.2}$$

$$velocity_i^{ts+1} = velocity_i^{ts} + \left(x_i^{ts} - x_{best}\right) Q_i \tag{3.3}$$

$$x_i^{ts+1} = x_i^{ts} + velocity_i^{ts+1} \tag{3.4}$$

In Equation 3.2, the (rnd) variable indicates a random vector that is randomly generated within the interval $[0, 1]$. This random vector controls the speed and ranges of the new generated velocity at specific time step by changing the value of Q_i which controls the output of Equation 3.3. This new velocity of the virtual bat n controls its new location using its current location at a new time step based on Equation 3.4. Here, the new location is referred to as the new current global solution (or the current global best).

The new current global solution is considered as the best global solution (x_*) which is to be compared in the next cycles of iteration. If there are improvements, the solution for bat n is then, considered as a new best global solution x_*.

The BA employs a local search approach (i.e. random walk) to improve the effectiveness and efficiency of its potential solutions using Equation 3.5. During the iteration of BA at time step ts, new solutions are locally selected based on random walk around the current best solution at that time step. If the new solution is better than the current best, then, the new best becomes a global best solution. These new local solutions are generated based on a random vector condition; the random vector at the time step ts for the bat n at cycle i must be greater than the pulse emission rate (r_i) for the associated bat. Mathematically, the random walk is defined as follows;

$$x_{new} = x_{best} + \epsilon A^{ts} \quad 3.5$$

In Equation 3.5, the symbol A^{ts} denotes the average of A_i^{ts} (i.e. the loudness of bats n at time step ts). The symbol ϵ drawn from $[-1, 1]$ is a random vector that controls the direction and strength of the random walk. To a certain extent, the BA is deemed to be a balanced combination of swarm optimization and the intensive local search governed by the frequency tuning ability, loudness variability, and pulse rate. Thus, for each iteration of BA, the loudness A_i and the emission pulse rate r_i are updated based on Equations 3.6 and 3.7.

$$A_i^{ts+1} = \alpha A_i^{ts} \quad 3.6$$

$$r_i^{ts} = r_i^0[1 - \exp(-\gamma\, ts)] \quad 3.7$$

Here, α and γ are BA constant variables that are having a similar effect like the cooling factor in a cooling schedule of SA algorithm in the range of $(0 < \alpha < 1)$ and $(\gamma > 0)$ with the exception demonstrated in Equation 3.8.

$$A_i^{ts} \to 0, \qquad r_i^{ts} \to r_i^0, \qquad ts \to \infty. \quad 3.8$$

The A_i^0 and r_i^0 are randomly chosen from $[0, 1]$ and the loudness A_i and the pulse emission rate r_i can only be updated when there is improvement in the new solution (i.e. the bats n are moving towards the optimal solution).

In this section, the BA is said to be overviewed. In the next section, the details of the BTS strategy and its algorithms that help the BA to construct *t*-way test suite are presented.

3.2 The BTS Strategy

Generally, the BTS strategy undergoes three phases during the construction of the mixed strength t-way test suite as shown in Figure 3.2. The three phases illustrated in Figure 3.2 processed as follows:

Phase 1. The BTS input analysis exploits the input analyser (parser) and legal values representation algorithm (see Algorithm 2). This phase set up the input for the next phase.

Phase 2. The BTS interaction generation which adopts two algorithms; CTS and IET generators (see Algorithm 3 and Algorithm 4, respectively), which are responsible of generating the required *t*-tuples.

Phase 3. The BTS test suite generation (see Algorithm 5) which exploits the BA as the core algorithm and exploits the Hamming distance selection criteria for mixed strength *t*-way test suite generation.

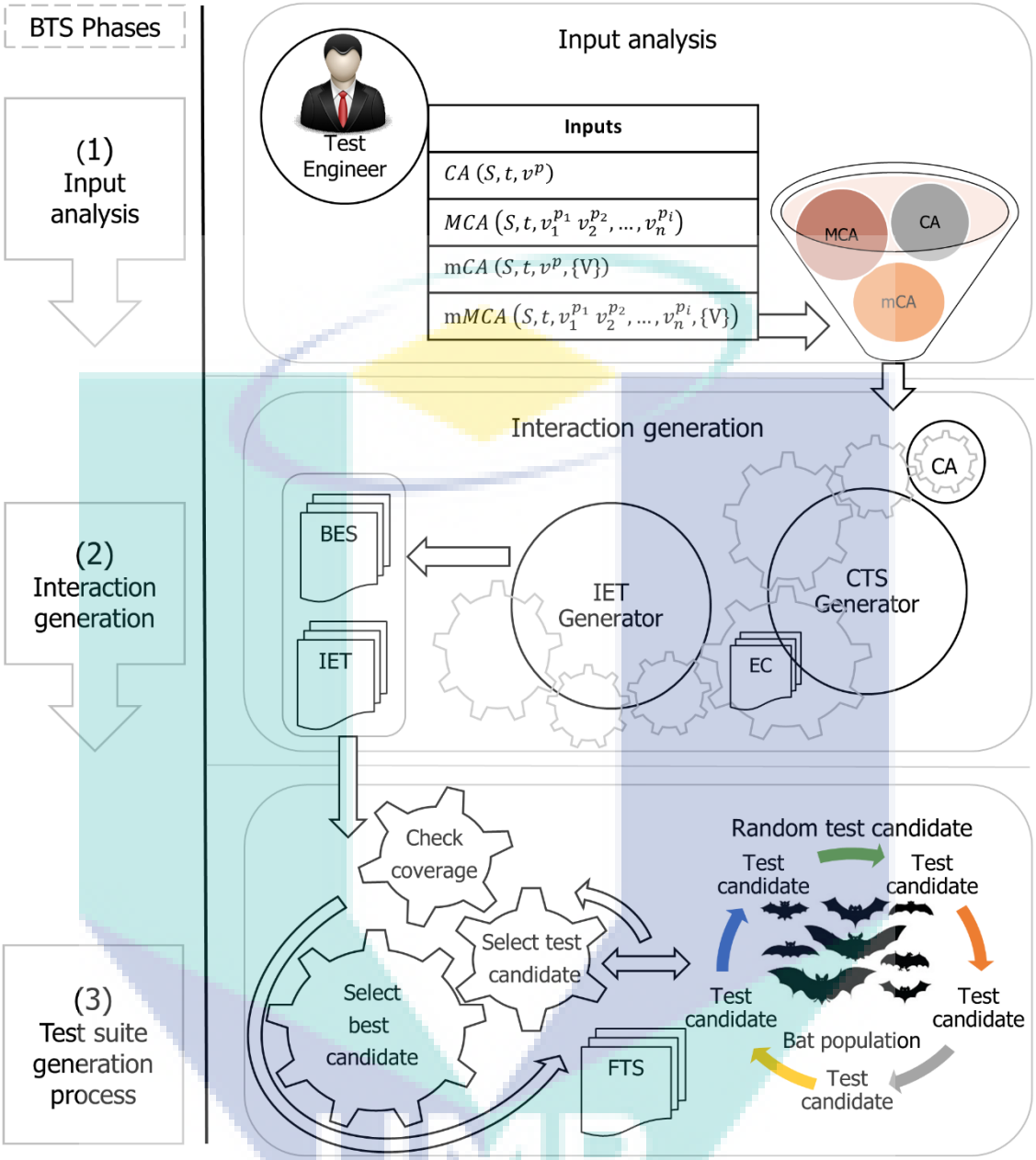


Figure 3.2 The overview of BTS strategy.

These phases and their algorithms will further discuss and elaborate in the next three sub-sections.

3.2.1 Input Analysis

In this phase, the input (CA, MCA, mCA or $mMCA$) processing is divided into two main processes; processing the input components (parameters and their values), and representing these components using numerical legal values. The first process starts by receiving the input, and then, processing the input components to a set of pre-defined variables in the memory (i.e. interaction strength (t), parameters (P) and their values (v)).

To clarify the BTS algorithms, consider the mixed-strength mixed covering array (i.e. $mMCA\ (S, 2, 2^2\ 3^2, \text{MC})$ where MC = $A\ (S, 3, 2^2\ 3^1)$ for the first three elements) as the running example from Chapter 2. In this $mMCA$ configuration, the interaction strength (t = 2) is considered for the overall system configuration. Then, the three-way (t = 3) sub-strength is considered based on its corresponding parameters. The overall test element sizes for each component are identified as test element sets (ES). Here, the element is representing the position of a parameter with it values, that called test component (i.e. the actual ES is ES = {E_1, E_2, E_3, E_4}). This ES is represented as a list in the memory, which follows the elements' indexes with their value sizes (i.e. ES = {2, 2, 3, 3} as E_1 = 2 values, E_2 = 2 values, E_3 = 3 values, E_4 = 3 values). Figure 3.3 shows the elements and their values construction on the ES.

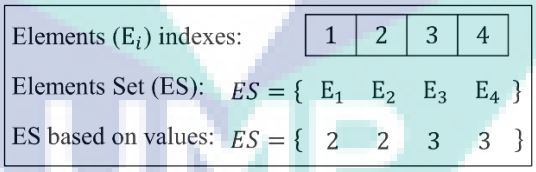


Figure 3.3 The illustration of the elements set based on the number of values for each elements.

The next process of the first phase is the representation of these components as Numerical Legal Values (NLV) (see Figure 3.4). These numerical legal values are processed and converted back to their actual values in the final test suite. The system is represented as follows:

- $E_1 = \{v_1, v_2\}$, represented numerically as [1]: 1, 2.
- $E_2 = \{v_1, v_2\}$, represented numerically as [2]: 1, 2.
- $E_3 = \{v_1, v_2\ v_3\}$, represented numerically as [3]: 1, 2, 3.
- $E_4 = \{v_1, v_2\ v_3\}$, represented numerically as [4]: 1, 2, 3.

```
Element set (ES) = {2,2,3,3}, Main-strength (t) = 2,
Number of Element: 4, Total Number of values: 10

Result of legal values representation:
Test Element [1][2 values]: [1, 2]
Test Element [2][2 values]: [1, 2]
Test Element [3][3 values]: [1, 2, 3]
Test Element [4][3 values]: [1, 2, 3]
```

Figure 3.4 The illustration of the variables processed in the input analysis phase.

Considering the sub-strength configurations, the three-way interaction for the first three test element is considered and processed as the following set; [sub-strength (t) : indexes of the E_i in ES]. Noticeably, BTS is supporting multi-degree sub-strength configurations (CA and MCA) based on the index of the E_i. The details of the sub-strength configuration representation can be seen in see Figure 3.5.

```
Element set (ES) = {2,2,3,3}

Sub-strength (mixed-strength) configurations:
[1] : [3:1,2,3] => (t = 3), element indexes: [1,2,3]
```

Figure 3.5 The illustration of the mixed-strength configrations

The complete processes in the input analysis is presented in Figure 3.6 (Algorithm 2, Input analyser and legal value representation).

Algorithm 2: Input analyser and legal value representation

Input: covering array notation (CA, MCA, mCA or mMCA).
Output: ES, LV, sub-strength specifications.

```
check the correctness of input;
define NLV, t, P, v;
define sub-strength, LV as two-dimensional set;
define E , ES as set;
process the main-strength system notation variables t, P, v;
If (sub- strength = true) then process the sub-strength setting each as single set,
then store each set in the sub-strength set;
for index (i) = 1 to P do
    assign P_i (index) to E_i;
    assign P_i (values size) to ES;
End
NLV = 1;
for each value in E_i do
    define line token (LT);
    LT_(index) = E_i;
    for i = 1 to ES_E_i do
        assign i to LT_i;
    End
    append LT to LV;
End
process results and visualization;
```

Figure 3.6 Pseudo code of input analyser and legal value representation algorithm.

The next section elaborates the generation algorithms for the combination elements and interaction elements are also used to handle the mixed-strength combination and interaction elements in the most efficient possible way. This process of generation algorithm reuse minimized the complexity of the generation process as well.

3.2.2 Interaction Generation

This phase involves the generation of the combination *t*-tuple sets (CTS), and the interaction element tuples (IET) based on the ES that contains the elements of the previously defined system-under-test. Here, the BTS employs a new generation approach that reversely generates the Binary Element Set (BES). The BES is important for the evaluation of the test cases coverage (the fitness of each test case provided by BA). This stage involves two algorithms; the CTS and IET generators.

The first algorithm (CTS generator) generates the *t*-tuple sets (CTS), and after completing the input analysis, a legal value (LV) will be generated. This legal value set is for the generation of the element combination (EC) which is the basis of CTS. The CTS generation starts as soon as it receives the legal value (LV) and the interaction strength (t) value from the input analysis phase. The construction of the EC sets uses multiple level pairing method to match all the possible element position references together based on the defined interaction strength.

Based on the $mMCA$ (running example) from the previous section, the pairing of the E_i in ES must take place for all the E_i based on the t =2. In this case, each two of the E_i selected together to construct the EC sets *(*see Figure 3.7). For the sub-strength, each set of the EC is constructed based on the specified *t=3*. In this example, only one EC set is constructed because there are only three E_i involved. The EC sets (generated pairs) for this example are as follows:

- The two-way element set pairing output for testing the overall system;

  $EC\ 1 = \{E_1, E_2\}$, represented numerically as EC [1] = 1, 2.

  $EC\ 2 = \{E_1, E_3\}$, represented numerically as EC [2] = 1, 3.

  $EC\ 3 = \{E_1, E_4\}$, represented numerically as EC [3] = 1, 4.

  $EC\ 4 = \{E_2, E_3\}$, represented numerically as EC [4] = 2, 3.

  $EC\ 5 = \{E_2, E_4\}$, represented numerically as EC [5] = 2, 4.

  $EC\ 6 = \{E_3, E_4\}$, represented numerically as EC [6] = 2, 4.

- The three-way element set pairing output for the sub-strength;

  $EC\ 7 = \{E_1, E_2, E_3\}$, represented numerically as EC [7] = 1, 2, 3.

All the represented ECs are then, stored in the CTS for further processing. For this example, there are six EC for the overall system and one for the mixed-strength in this case (Figure 3.7). The complete steps of the CTS generator can be seen in Algorithm 3 (Figure 3.8).

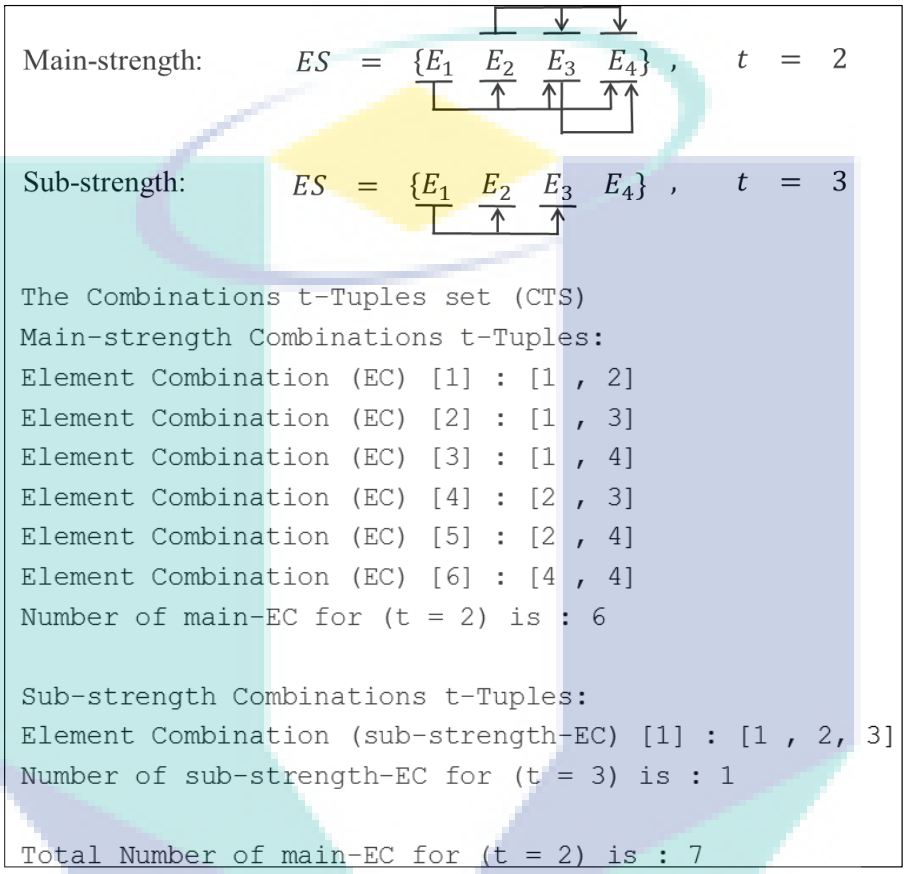


Figure 3.7 The illustration of EC matching and EC in CTS.

Here, the CTS generator (Algorithm 3) has identified and constructed all the EC (also known as *t* based pairs) for the involved interaction strength (the 2-way main-strength and the 3-way sub-strength). For the 3-way sub-strength, there is only one EC in this case; however, in the case of more E_i in the ES involving 3-way, there will be more ECs. For instance, if the 3-way sub-strength are assumed to be involved in all E_i in ES, there will be four ECs. The generated EC sets in the CTS serve the purpose of covering all the *t*-tuple of the software-under-test. As in the next stage of this phase, a binary element for each EC set is going to be constructed for the coverage calculation.

Algorithm 3: CTS generator

Input: t, ES, sub-strength specification set.
Output: CTS.

```
1:  define EC, CTS as empty set;
2:  for each E_i in ES do
3:      construct EC by matching E_i with other E_i in ES based on t degree;
4:      append EC to CTS;
5:  End
6:  for each sub-strength specification in sub-strength do
7:      get sub t value;
8:      get Indexes for sub-strength E_i;
9:      define sub-strength-EC as empty set;
10:     for each (E_index)in Indexes do
11:        select E_i for E_index;
12:        append E_i to sub-strength-EC;
13:     End
14:     if (sub-strength-EC length = sub t)
15:     append sub-strength-EC to CTS;
16:     Else
17:         for each E_i in sub-strength-EC do
18:             construct sub-strength-EC by matching E_i with other E_i in ES based
                on sub t degree;
19:             append sub-strength-EC to CTS;
20:         End
21:     End
22: End
23: process results and visualization;
```

Figure 3.8 The pseudo code of CTS generator.

The second stage of this phase is to construct all the interaction elements tuples based on the represented binary element for all the EC in CTS. The two processes of generating IET and BES are combined in one algorithm to reduce the complexity of the generation process (i.e. the generation of all the possible binary elements and the selection of the binary elements that seems to be inefficient when done separately).

The IET generation starts with the selection of all the EC in the CTS and traversing them until all the interaction elements (IE) are generated for each EC. In the $mMCA$ example, there are seven EC sets that are specified based on E_i. The EC sets are requested to generate the IE elements numerically based on the numerical legal values represented earlier. The matching approach similar to the construction of EC in CTS was adopted in this process. The only difference is that in IET generation, the values of the E_i in each EC were matched. For example, the first EC in the CTS involves E_1 and E_2, these elements are represented in the ES set as the first and second indexes (see Figure 3.9).

Next, the value of elements E_1 and E_2 are called from the legal value set, which in this case, two values per element are involved (i.e. 1 and 2, referring to v_1 and v_2, respectively). These values are cross-matched together to construct the representative IE sets. In other words, the values for the involved elements interacted together to construct the IE sets. In this case, E_1 consists of two values ($v_{E_i} = 2$) and similar values can be seen for E_2. Thus, the number of IE sets for this EC can be calculated using Equation 2.7, yielding $2 \times 2 = 4$.

The process of matching the involved E_i values is done one IE at-a-time (row by row), with the IEs having the same ES index length. The values involved are added to their represented E_i indexes and the "*x*" (Don't care values) are represented numerically as "-1". For each EC, one binary element (BE) representation with the same ES length is constructed (i.e. the BE bit length equals ES bit length), and the values of the involved E_i in the specified EC are represented as "1", and the "*x*" values as "0". The details of this process are illustrated in Figure 3.9. The BE element is then, stored in the BES list for further usage in the next phase.

This process is sustained throughout all the EC until the six 2-way sets of IEs are generated, then combined together into a single IET. The IET here, contains six sets of IEs. The BES contains the BE representation of the six 2-way EC that generates the six sets of IEs. In a same manner, the same process goes also for the EC that are constructed from the sub-strength. The BE and IE for the mixed-strength configurations consists of one IE set with three interacted E_i which also added to an IET (i.e. a one BE element to represent the single EC for the 3-way sub-strength).

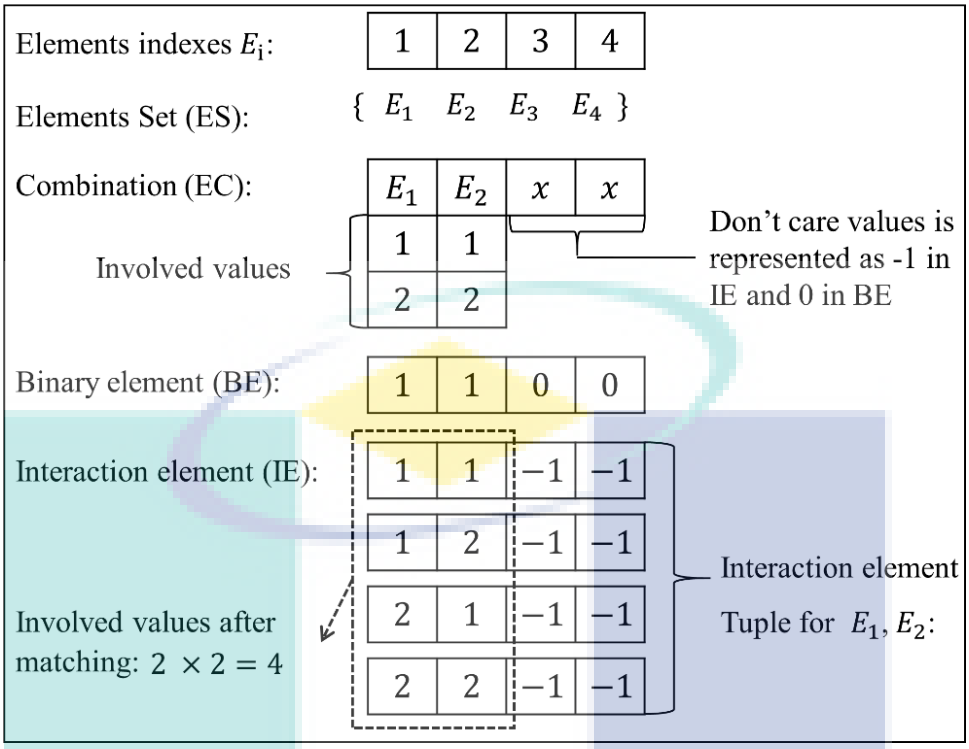


Figure 3.9 The construction of interaction elements and binary elements.

In the current example, there are seven groups of constructed IE sets with each set having a number of IE rows. The number of IE elements in all the sets based on the Equation 2.7 is as follows:

- The two-way IE's output for the main-strength;

  IE set 1 involves EC 1 = $\{E_1 \times E_2\}$ → 2 × 2 = 4 IEs.

  IE set 2 involves EC 2 = $\{E_1 \times E_3\}$ → 2 × 3 = 6 IEs.

  IE set 3 involves EC 3 = $\{E_1 \times E_4\}$ → 2 × 3 = 6 IEs.

  IE set 4 involves EC 4 = $\{E_2 \times E_3\}$ → 2 × 3 = 6 IEs.

  IE set 5 involves EC 5 = $\{E_2 \times E_4\}$ → 2 × 3 = 6 IEs.

  IE set 6 involves EC 6 = $\{E_3 \times E_4\}$ → 3 × 3 = 9 IEs.

- The three-way element set pairing output for the sub-strength;

  For IE set 7, EC 7 = $\{E_1 \times E_2 \times E_3\}$ → 2 × 2 × 3 = 12 IEs.

The total interaction elements are 49 generated IEs which can be also calculated using Equation 2.5. Here, these 49 IEs are the sum of all the IE sets from the two groups of main and sub-strength. The details of the full IE sets in IET are shown in Figure 3.11.

In this phase, the generation process constructs a dynamic multi-dimensional IET set which has all the IE sub-sets for each EC in CTS. The use of dynamic multi-dimensional sets in BTS is necessary to avoid the limitations of the set index limit (set by the programming language). As the BTS is designed to fit large number of test suite generation, the concept of dynamic multi-dimensional set allowing BTS to store each IE set in a bigger set called, IET. In this manner, the IET set does not need any indexing (see Figure 3.10).

Algorithm 4: IET generator	
Input: t, CTS. **Output**: IET, BES.	
1:	define IET as multi-dimensional empty set;
2:	define BES as empty set;
3:	define IE as empty set;
4:	**for** each EC in CTS **do**
5:	get the involved v_{E_i} from LV that represent E_i for EC;
6:	define Temp as empty set;
7:	construct IE by matching v_{E_i} with other v_{E_i} in LV based on the t degree;
8:	append IE to IET;
9	construct BE for EC based on the ES index length;
10	append BE to BES;
11	**End**
12	process results and visualization;

Figure 3.10 The pseudo code of IET generator that includes BES generation method.

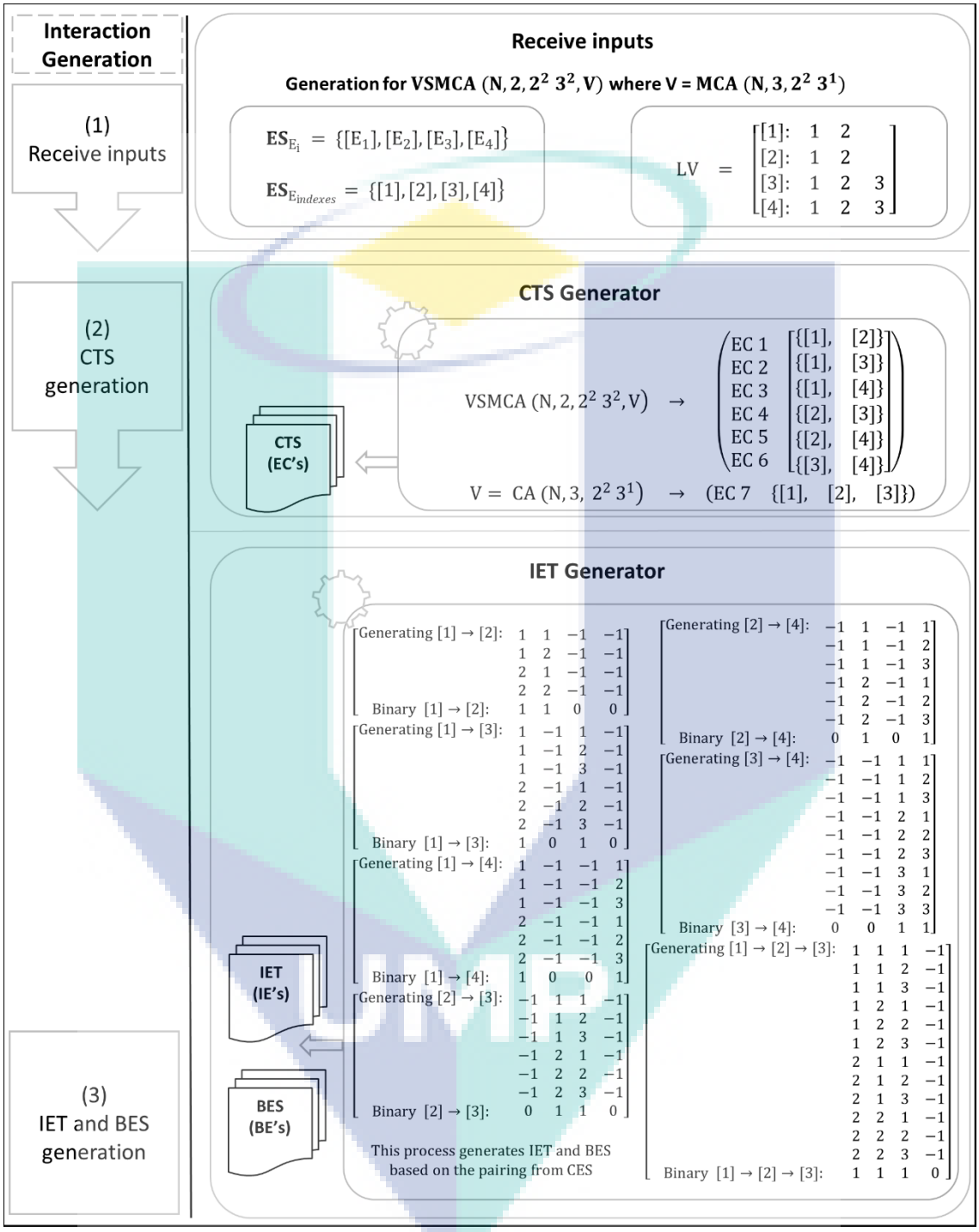


Figure 3.11 The illustration of the CTS, IET and BES generation flow.

Another important point in this generation method is the construction of the BE (or binary equivalence) to represent each EC. The BE is important for the determination of the test case coverage. The generation of the binary equivalence for each EC sets is constructed using the same loop during the generation of the IEs. The BE and IE sets are then, appended to each corresponding set. Unlike the traditional approaches such as the GTWay (Zamli et al., 2011) where the BEs were generated before the IEs, the current approach generates the BE in the later part of the process. The rationale for such approach is the fact that traditional approaches of generating full table of binary equivalences has a high complexity in case of systems with a high number of components (large number elements of the test generation targeted domain). This is because it follows a $(2^E - 1)$ number of elements.

In the BTS strategy, the mixed-strength interaction generation is combined with the main process to minimize the algorithm's complexities. A sub ES set for each mixed-strength configurations constructed based on the indexes of the involved test elements was used in this study. The same CTS and IET generator can be used to generate the mixed-strength CTS and IET set respectively.

At the end of this phase, the aforementioned steps effectively generate complete sets of IEs that covers all the possible test interactions. These IEs sets needed to be optimized to achieve the test suite reduction. In the next sub-section, the reduction process based on the optimization concepts of BA is elaborated.

3.2.3 Test Suite Generation

The BA can be used to efficiently solve several related engineering optimization problems (Yang, 2010, 2014; Yang & Gandomi, 2012). This algorithm can improve the solution quality because of the global and local search behaviour it implements. Here, the BA is employed as a search engine to calculate the fitness (*coverage* or *weight*) of the randomly generated test candidates for the proposed strategy. To achieve a minimum test suite optimization process, the test cases need to effectively and greedily cover all the *t*-way tuples, if possible, at most once.

The BA has conventionally developed on the assumption that bats can locate their prey in complete darkness. To apply this algorithm for interaction testing, we assume that the test candidates are bat locations in which each bat has its own possible solution (*fitness*) of the targeted problem. The BA search process in the BTS provides the best global optimum (or optimum test candidate that has the highest coverage of the *t*-tuples element values) based on the number of BEs involved. This optimum test candidate indicates the solution quality in terms of the best bat position (*location*) from its prey. Bats are avoiding obstacles using echolocation, thus, different frequencies are returned in each iteration with updated loudness and pulse emission rate.

Based on the aforementioned description of the BA, Figure 3.12 (Algorithm 5) depicts the complete algorithm as the backbone for BTS. Unlike the standard BA algorithm, the BTS strategy introduces the Hamming distance classifier to decide the final suite. Specifically, the Hamming distance classifier measure two rows of (best) test cases (as string) based on the number of values in which they differ when there is a tie situation as far as the quality of the test cases are concerned. It is the farthest test case that will be finally selected by the Hamming distance classifier to ensure sufficient exploration of the search space.

Lines 1 - 5 represent the initialization process which includes the request for the pre-processed data from the previous algorithms in the interaction generation phase (the interaction elements tuples (IET)), and the binary elements set (BES) (constructed based on algorithm 2 and 3 in the second phase of BTS strategy).

Algorithm 5: BTS test suite generation.

Input: BES, IET, *objective function* $f(x_i), x_i = (v_{E_1}, \dots, v_{E_i})$.

Output: FTS

request BES, IET and the objective function specification;
define FTS as empty set;
define x_{best} best test candidate;
initialize BA variables setting(n, T_{max}, $Q_i \in (Q_{min}\ ,\ Q_{max})$, A, r, $tolerance$ and ts)
randomly Initialize BTS population(n_i, x_i, v_i, f_i, Q_i)for $i = 1, 2, \dots\ ,\ n$;
evaluate initial population;
select x_{best} from all x_i^{ts} in BTS population; \\(initial global best)
while (IET is not empty) **do**
    while ($ts < T_{max}$) **do**
        for each bat n_i in the population **do**
            generate new test candidates using motion movement equations;
                $Q_i = Q_{min} + (Q_{max} + Q_{min})\ rand(0,1)$, \\ $rand \in (0,1)$
                $v_i^{ts+1} = v_i^{ts} + \left(x_i^{ts} - x_{best}\right) Q_i$
                $x_i^{ts+1} = x_i^{ts} + v_i^{ts+1}$
            re-evaluate the population;
            if ($rand(0,1) < r_i$) **then**
                if ($f\left(x_i^{ts+1}\right) > f(x_{best}\ )$) **than** $x_{best} = x_i^{ts+1}$;
                (new global best from current population)
            Else
                generate a new x_i^{ts+1} by flying randomly around x_{best};
            End
        End
        for each bat n_i **do** \\ random walk
            if ($rand(0,1) < A_i$ and $f(x_i^{ts+1}) > f(x_{best})$) **then**
                accept the new x_i^{ts+1} as new x_{best}; \\(new global best)
                increase r_i and reduce A_i using the $tolerance$ value;
            End
        End
        for each bat n_i **do** \\ Hamming distance classifier
            if (ts not equal 1 && multiple x_{best}) **then**
                select x_{best} in the current population that has the highest distance from x_{best} added to FTS;
            End
        End
    End
append x_{best} to FTS;
remove the covered IE's by x_{best} from IET;
End
process results and visualization;

Figure 3.12 The pseudo code of test suite generation.

Specifically, line 2 defines a new set for BTS output in the initialization process called the Final Test Suite set (FTS). In addition, the global best test candidate is also defined in line 3, which are going to store the best test candidate during the iteration of the algorithm. In line 4, the BA variables (settings); bats population size (n), BA iteration (T_{max}), initial loudness (A_0), initial emission of pulse rate (r_0), tolerance and frequencies range (i.e. minimum frequency (Q_{min}), maximum frequency (Q_{max})) are initialized in addition to the time step counter (ts).

In line 5, the BTS modelling the test candidates as bats location ($x_{bat\ i}$). Here, the population is initialized based on the assumption that bats location ($x_{bat\ i}$) are the test candidates. The population is constructed as sets with a number of bats (n_i), each having its owns location, velocity, fitness and frequency. The location of each bat are randomly initialized as a test candidate constructed based on the decision values (v_{E_i}) that represent the test element in the ES. Each location represents a random test candidate ($x_{bat\ i}$) that is indexed for a specified bat as shown in Figure 3.13.

Until now, the BTS iterations are yet to be started. The velocity and fitness for all the bats in the population are still having an initial value of zero. In contrast, the frequency (Q_i) is generated randomly in the range of Q_{min} to Q_{max} for the first iteration as the frequency is adjusted in the next cycle of iterations. Likewise, the velocity and location are updated for each bat in every cycle of iteration using the virtual bat movement (based motion equations).

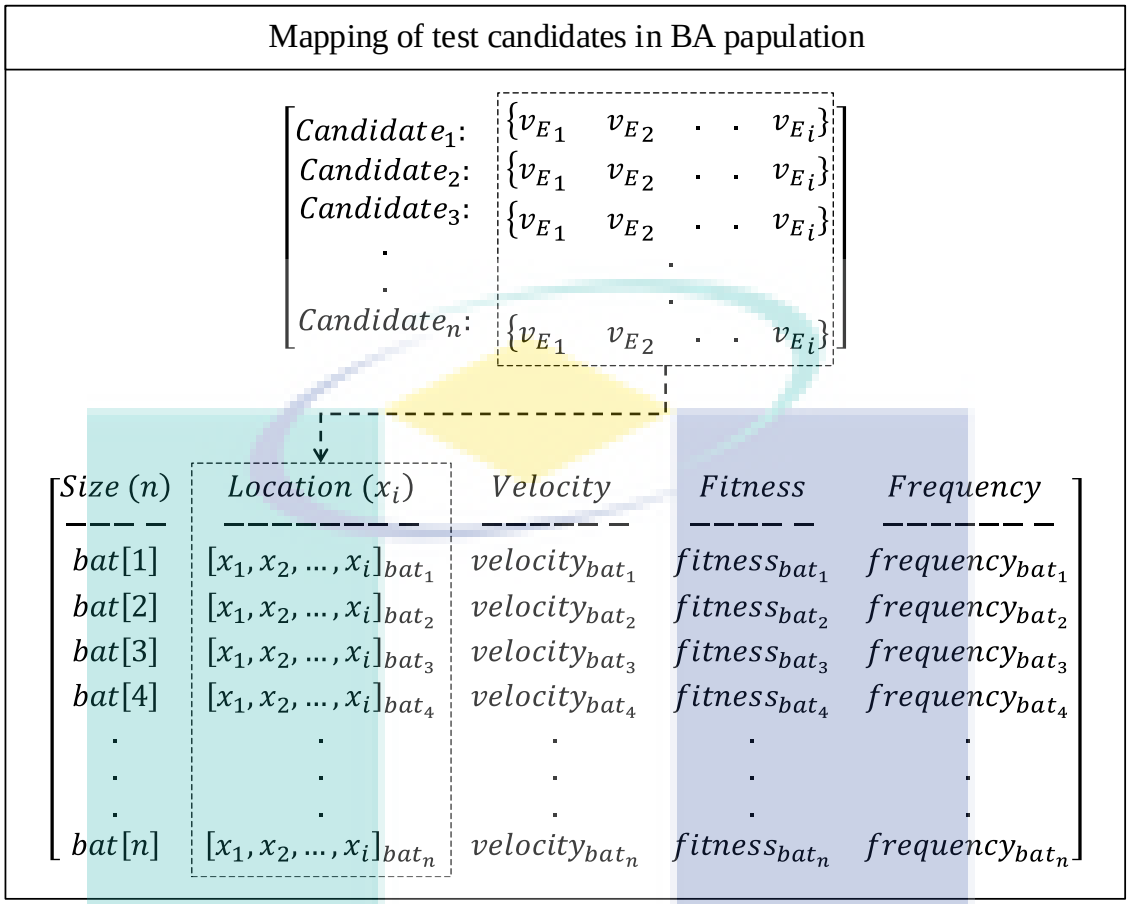


Figure 3.13 The illustration of test candidates mapping into the BA population.

In lines 6 and 7, the initial evaluation (or the evaluation of the first generation) and the selection of the initial global best test candidate is processed. The evaluation steps calculate the fitness of all the bats in the population and store the values in the corresponding fitness field (*column*) for each specified bat in the population (Figure 3.13). Here, the test candidate ($x_{bat\,i}$) is evaluated using the objective function specified earlier in Equation 2.8. The fitness is presented as the number of IEs covered by each bat. The best test candidate with the highest coverage value (or *fitness*) is then, is selected and considered as the best global bat.

To select a best global coverage in case of multiple test candidates with the same maximum coverage and empty FTS, the Hamming distance classifier was not used in this step to determine the bat having the highest distance from the test cases in FTS as FTS is still empty. The first bats with the highest coverage values are selected randomly as the best global bat as they have the same maximum coverage value.

As the global best is selected, the iteration is initiated in line 8 where the BTS cycle until the IET, is empty. This consider as the stopping condition because all the test cases that covered all the IEs in the IET is generated. During this iteration from line 8 to line 33, the iteration for the BA time steps is cycling for all the bats as represented in line 10 to line 30. The bats here are improved in each time steps or population generation.

The process of improvement is progressing in line 11 and each bat solution is updated based on the virtual bat movement (motion movement) using Equations 3.2 to 3.4. The improvement process of each test candidate is as follows: firstly, a new frequency is calculated for the specified bat using Equation 3.2 based on a random variable (*rand*) within the range of zero to one; secondly, a new velocity is calculated based on the current best global test candidate (the best test candidate selected earlier), and the new frequency using Equation 3.2; finally, a new test candidate is generated based on the new calculated velocity and the current test candidate in the population using Equation 3.3.

In line 12, the new test candidate is generated using the new velocity for the targeted bat. A re-evaluation process of the generated solutions based on the objective function is carried out (i.e. owing to the potential improvements).

Lines 13 – 17 update the test candidate using a random variable based on the emission of pulse rate values. If the random is less than the emitted pulse rate, and the fitness of the current bat is greater than the current global best, the test candidate considers as a global best. In order words, a random walk (local search) for the current test candidate will be considered. During the local search, the bat's location (test candidate) is randomly generated around the best global candidate.

Concerning the selection process in lines 19 to 24, a random variable is generated but if it is less than the current loudness and the fitness of the current test candidate is higher than the best global test candidate fitness (i.e. ($x_{bat\ i}$) is covering more interaction elements), then the test candidate is considered as the new global best. The algorithm then, increases the rate of pulse emission and reduces the loudness using the tolerance value. This case selects the best test candidate. In the case where multiple test candidates have similar maximum fitness, the Hamming distance classifier is called to decide which test candidate should be selected as the global best.

Lines 25 – 29 represent the updating of the best global test candidate based on the Hamming distance classifier with multiple test candidates with the same maximum coverage value. The generation step improvement is not equalled to one (i.e. not in the first cycle, as in this case the FTS is still empty). Here, two or more test candidates can share multiple best with the highest fitness. To break the tie situation, the Hamming distance classifier is adopted to select the candidate that has the farthest distance among the best candidates with respect to the test cases in the FTS using Equations 3.9. The distance term calculated for the test candidates with all the available test cases in FTS are shown in Figure 3.14.

$$Hamming\ distance\ (d(x)) = \sum \left| x_{bat_i} - TC_n \right| \qquad 3.9$$

Let us assume the test candidate $x_{bat\ 2}$ = {2 1 3 2} and the available test case TC_1 = {1 1 1 1}, here, $d\ (x_{bat\ 2}, TC_1)$ = 4; the next test case TC_2 = {2 1 1 3}, $d\ (x_{bat\ 2}, TC_2)$= 3, and so on. In all these, the $d\ (x_{bat\ 2}) = 7$. This process goes through all the best candidates (i.e., $d\ (x_{bat\ 4}) = 5$ and $d\ (x_{bat\ 6}) = 9$). In this example, the best test candidate with the highest (or *farthest*) distance (which is in this case $x_{bat\ 6}$) is selected accordingly (refer to Figure 3.14).

The iteration of the algorithm is contained until it satisfies the number of generations (T_{max}) specified. In each cycle of generation, the test candidate is improved until the maximum coverage is achieved. The final best global candidate is considered as final test case that is added to the FTS. Then, the covered IEs are removed from the IET as shown in line 32. This process elaborated above is maintained until all the IEs in the IET has been covered (IET became empty). The final test suite is then, displayed.

It should be noted that when there is only one best candidate satisfying the maximum coverage, the Hamming distance classifier will not be considered (lines 15 to 18). But in the case of multiple bats with the same maximum coverage occurred, the Hamming distance classifier is used to determine the best candidate that need to be selected.

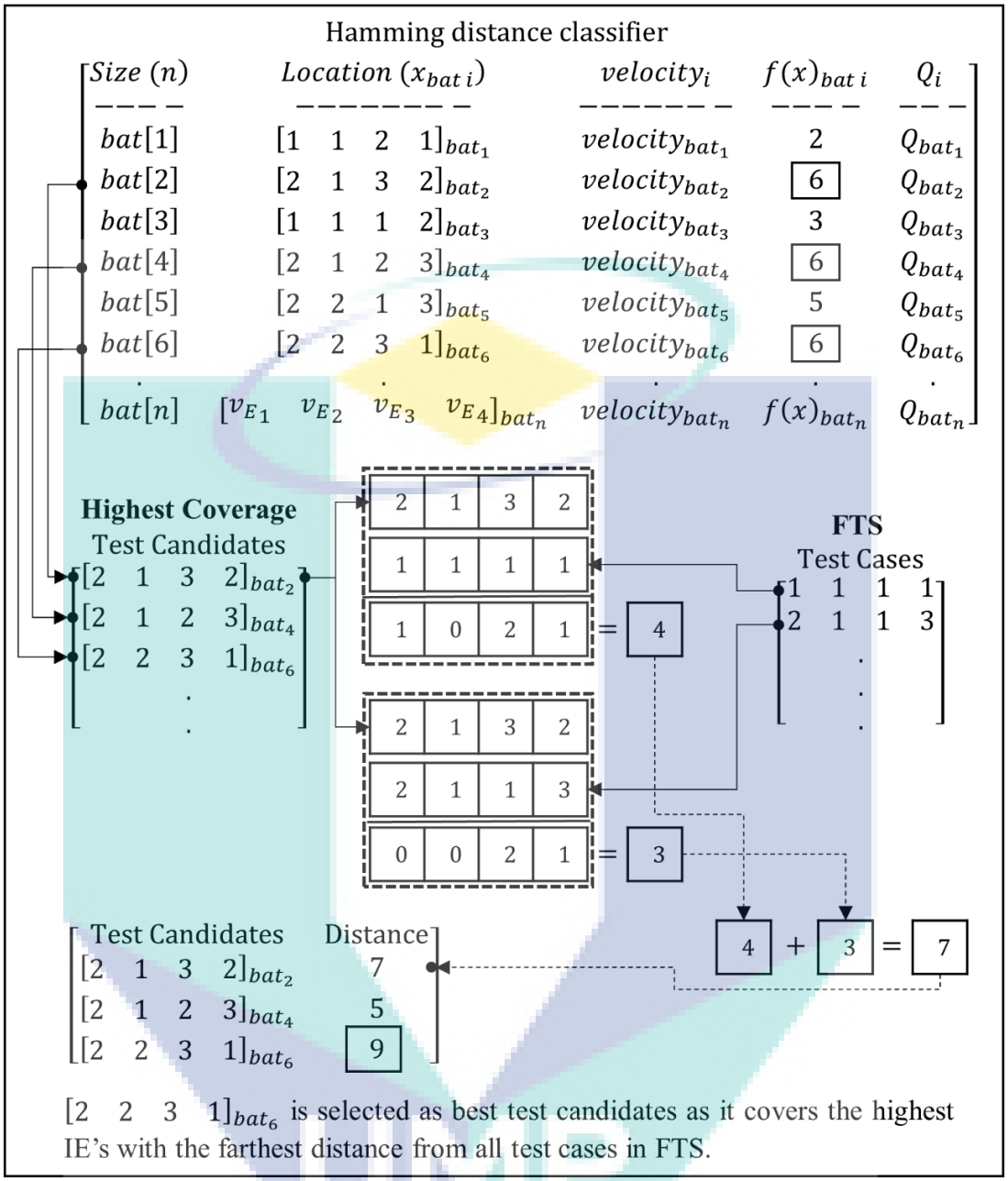


Figure 3.14 The illustration of Hamming distance classifier.

In a situation where the BA iteration (T_{max}) has completed and the best bat has not satisfied the maximum coverage (there are still elements in the IET), the test candidate will be appended to the FTS. Here, the improvement process in the BTS failed to find the minimal test suite.

The test suite generation algorithm stays in the cycling until the exit criteria are met. When the exit criteria are met (all the involved IEs in the IET are covered) the cycling stops and the FTS will be displayed. But on the contrary, the update and improvement of the test candidates will be going on until all the IEs in the IEL are covered.

In the case of mixed-strength (sub-strength), the objective function is evaluated for its EC and the same process is undertaken. The obtained test suite for the main-strength and the sub-strength are combined without the duplicates from the test cases in the sub-strength test suite.

For more details, Figure 3.15 shows a simple illustration of the mechanism (or *flow*) of the of test suite generation in BTS.

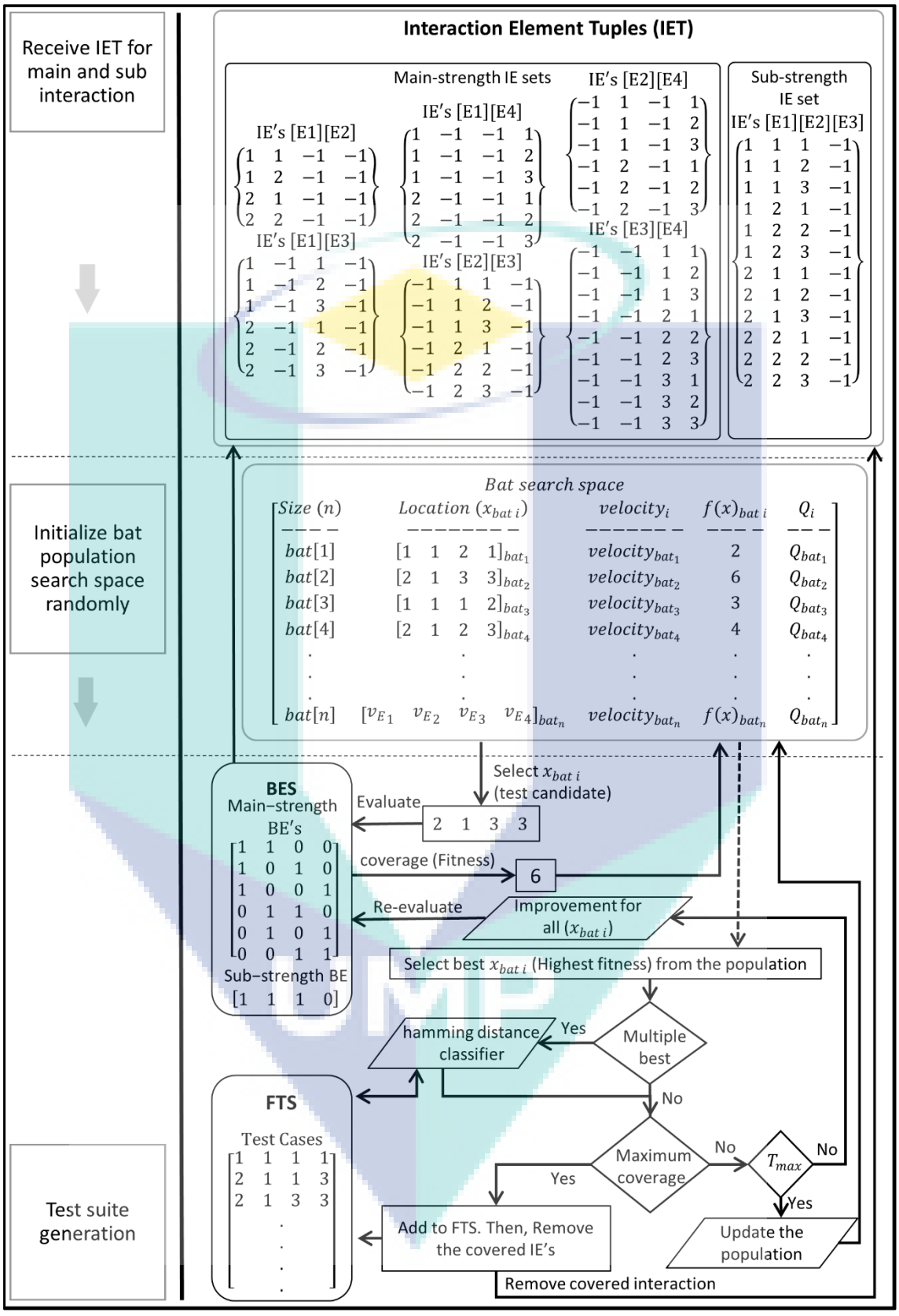


Figure 3.15 The illustration of test suite generation mechanism.

3.2.4 Tuning of BTS Variables

The process of the variables tuning for the BTS has been elaborated in this sub-section. To ensure the most optimal results for the BTS with regards to the test suite size, the control variables for the BA have to be tuned based on the test suite generation problem. For the tuning purpose, a well-known test system (covering array) that involving a CA (N; 2, 5^7) is employed. The justification for adopting this configuration for the tuning process originates from the use of the same CA to tune many of the existing *t*-way strategies (Ahmed & Zamli, 2011b; Alsewari & Zamli, 2012a; Stardom, 2001). The process of tuning BTS is based on 20 runs (Ahmed & Zamli, 2011b; Stardom, 2001) for the specified CA with different variables settings. The BTS has five main control variables: {bats population size, generation, loudness, pulse rate, and tolerance}, that control the sizes of the obtained test suite. The size and average of the final test suite sets for the 20 runs have been recorded. Then, the results of the tuned variables are analysed to find the settings that fit the minimum size and average of the final test suites (Figure 3.16 to Figure 3.23) (refer to Appendix B for a full detail of the variable setting execution outputs). The five variables (bat population and generation (iteration), loudness, pulse rate, and tolerance are executed for all the possible selected settings.

The BA setting variables are varied depending on the problem to be solved. For instance, Yang (2014) has used the following parameters; {n = 25 to 50, T_{max} = 10 , A_i = 0.25, r_i = 0.5, Q_{min} = 0, Q_{max} = 2, tolerance = 0.00001}as a default setting values for his evaluation, while Yang and Gandomi validated BA for solving a global engineering optimization problems using a fixed n = 20 and g = 1000 to perform 20 thousand searches which was not necessary for testing the suite generation (Yang & Gandomi, 2012).

Yang (2014) also specified that if the loudness and pulse rate are fixed to 0 and 1, respectively, the BA can behave as the standard PSO in this setting. On the other hand, if these settings are fixed in the range of 0.7 to 0.9 (i.e. A_i = r_i from 0.7 to 0.9), the BA basically becomes a HS as Q_i change is equivalent to the pitch adjustment in HS. In another study, researchers used n = 20, A_i = 0.9, r_i = 0.1, Q_{min} = 0, Q_{max} = 2 for their evaluation of a combined economic load and emission dispatch (Rakesh et al., 2013). The BA frequency variables were tuned in this process in a pre-specified range; the tolerance value was pre-defined for local search use as well.

The first variable is bat population (n) and the iterations; for the bat population which indicates the number of bats involved in the test suite generation, this variable controls the randomly initialized test candidates in the memory. When the number of test candidate increases, the possibility of finding a better solution (a test candidate with maximum IEs coverage) improves and vice versa. Yang (2010) specified that the number of bat population can be more than 10 depending on the problem solved. Yang (2010) also set the bat population from 10 to 40 in his simulations for finding the global optima for several benchmarking functions (Yang, 2010). For the tuning of bat population, a set of four values (n = {10, 20, 50, 100}) were selected.

Next, the iteration (generations (T_{max})), as name suggests, controls the improvement of the solution for all the initialized bats (*test candidates*) for each cycle of iteration. It is noticed that when the iterations variable increases, the possibility of founding a better solution is increased and vice versa. For the tuning of the maximum generation, we selected a set of five values (T_{max} = {10, 20, 50, 100, 200}). These bat population and generation values appear effective for obtaining the minimum test suite. Our experiments' shows that the quality of the BTS solution is acceptable for these settings. Thus, these sets were used for a full tuning execution.

The initial values of the loudness and pulse rate need to be tuned; the maximum effectiveness of these variable are in the range of [0 – 1]. Thus, a set of five values are selected in this range for both variables. The set contains five values (i.e. A, r = {0.05, 0.25, 0.5, 0.75, 0.95}). For the tolerance that controls the adjustment of the loudness and pulse rate, another five values (i.e. tolerance = {0.00001, 0.0001, 0.001, 0.01, 0.1}) are selected to fit the test suite generation.

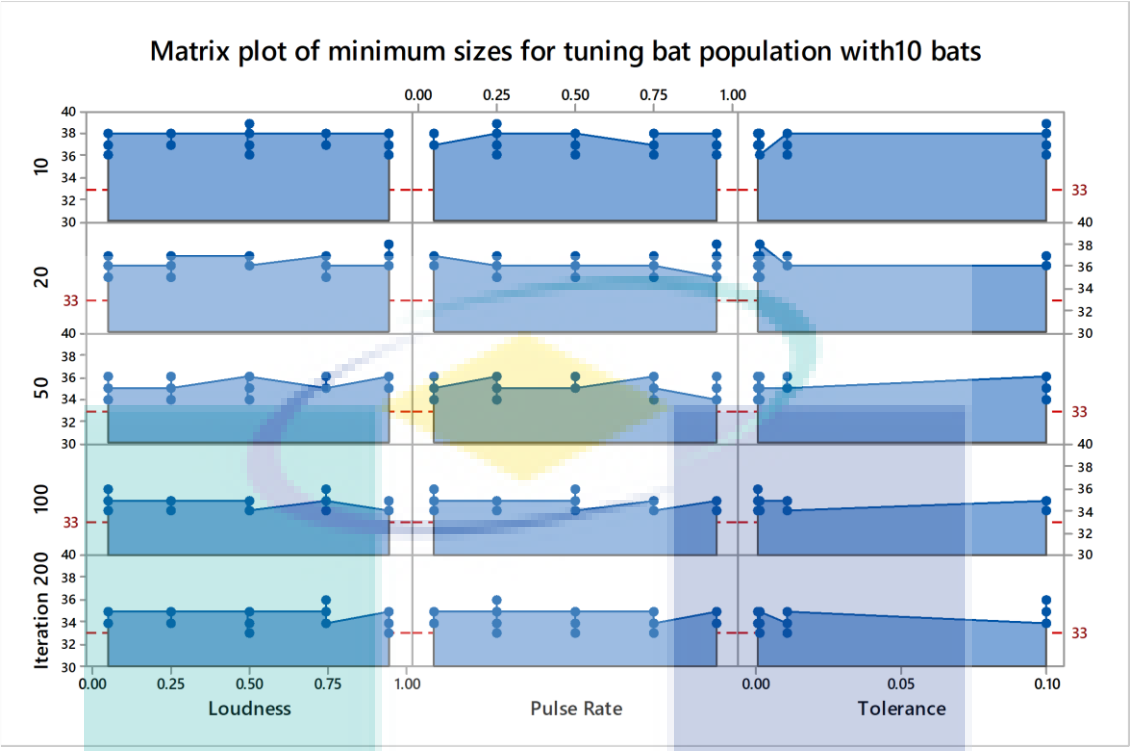


Figure 3.16 The illustration of minimum sizes with 10 bats.

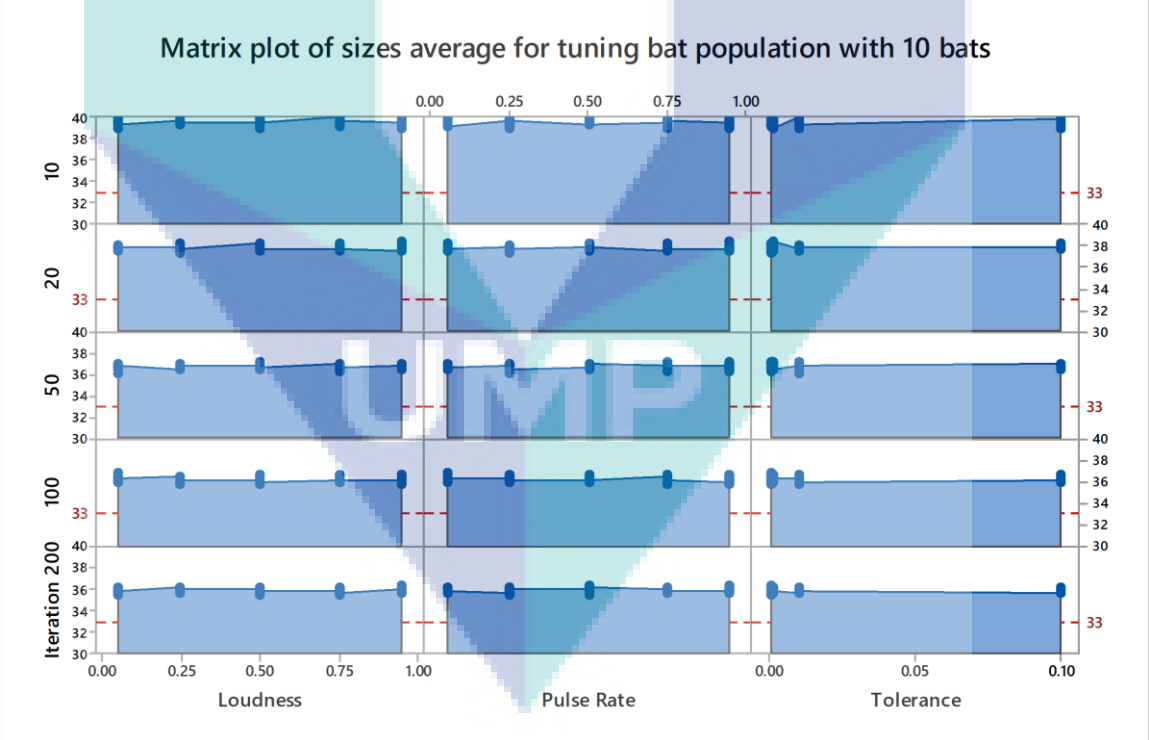


Figure 3.17 The illustration of sizes average with 10 bats.

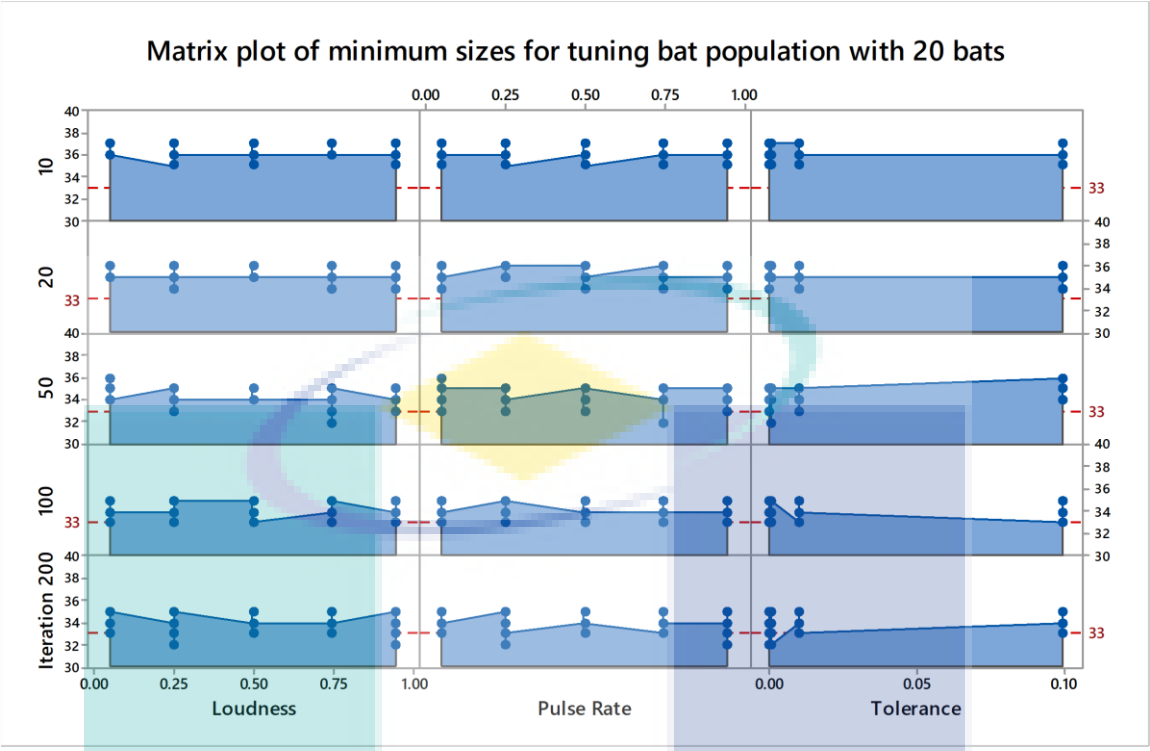


Figure 3.18 The illustration of minimum sizes with 20 bats.

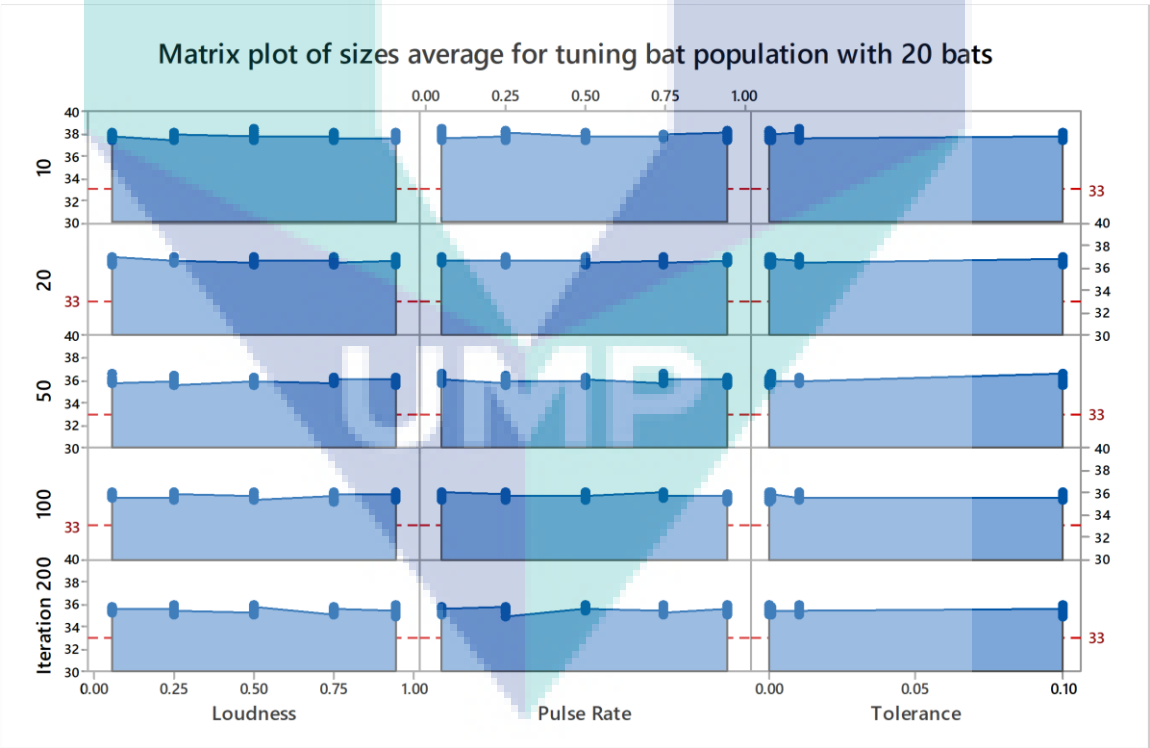


Figure 3.19 The illustration of sizes average with 20 bats.

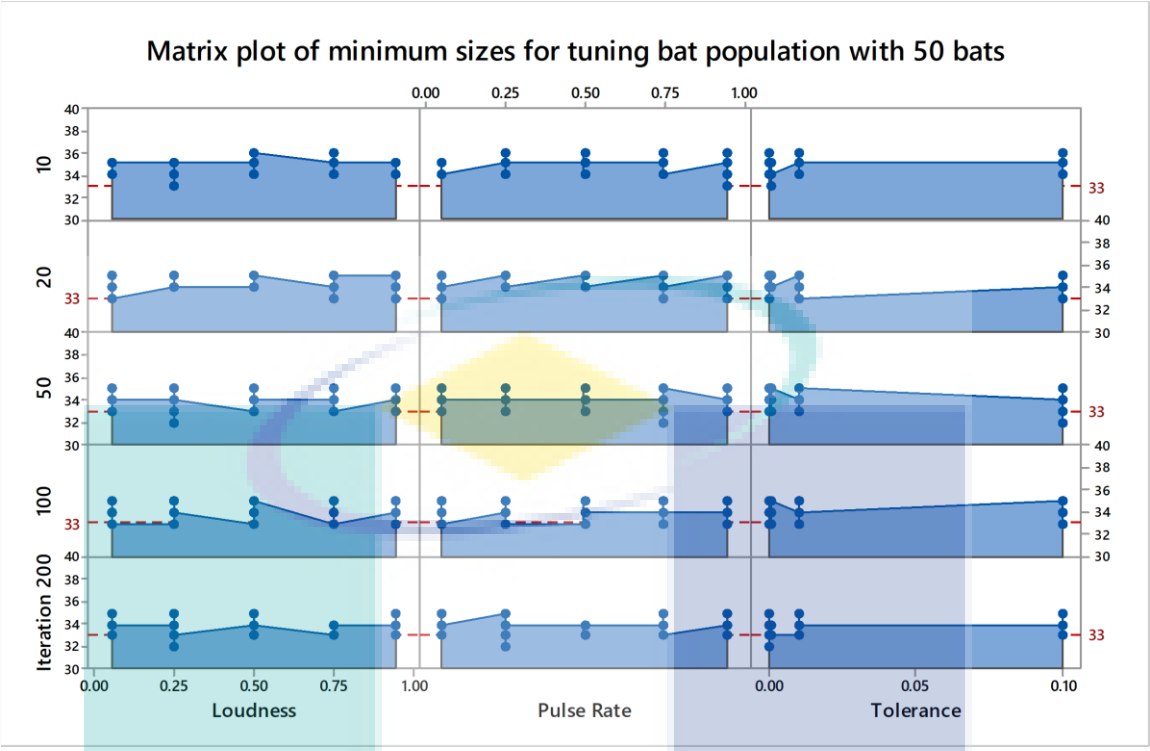


Figure 3.20 The illustration of minimum sizes with 50 bats.

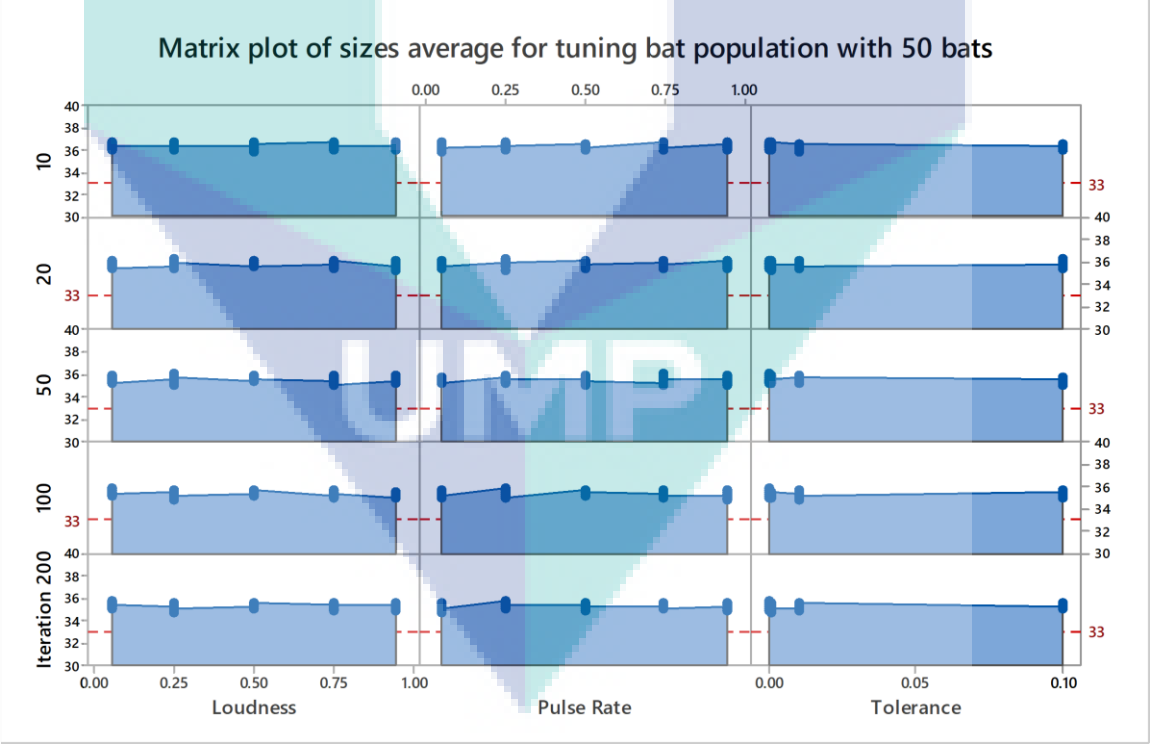


Figure 3.21 The illustration of sizes average with 50 bats.

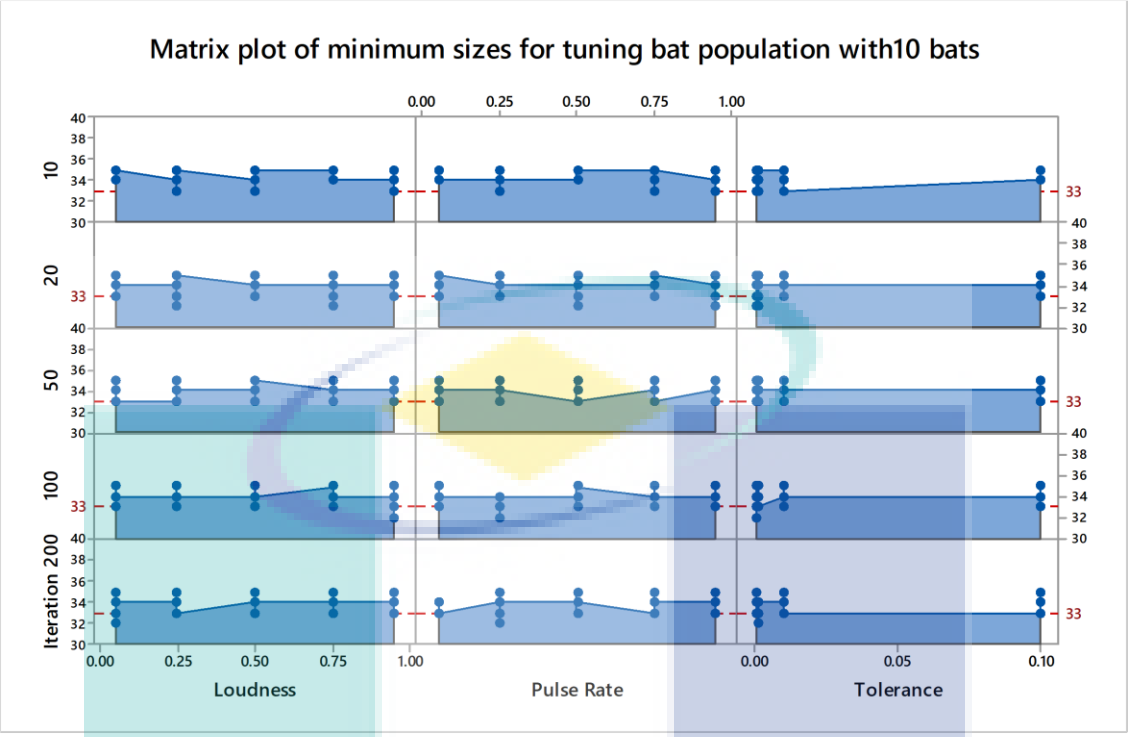


Figure 3.22 The illustration of minimum sizes with 100 bats.

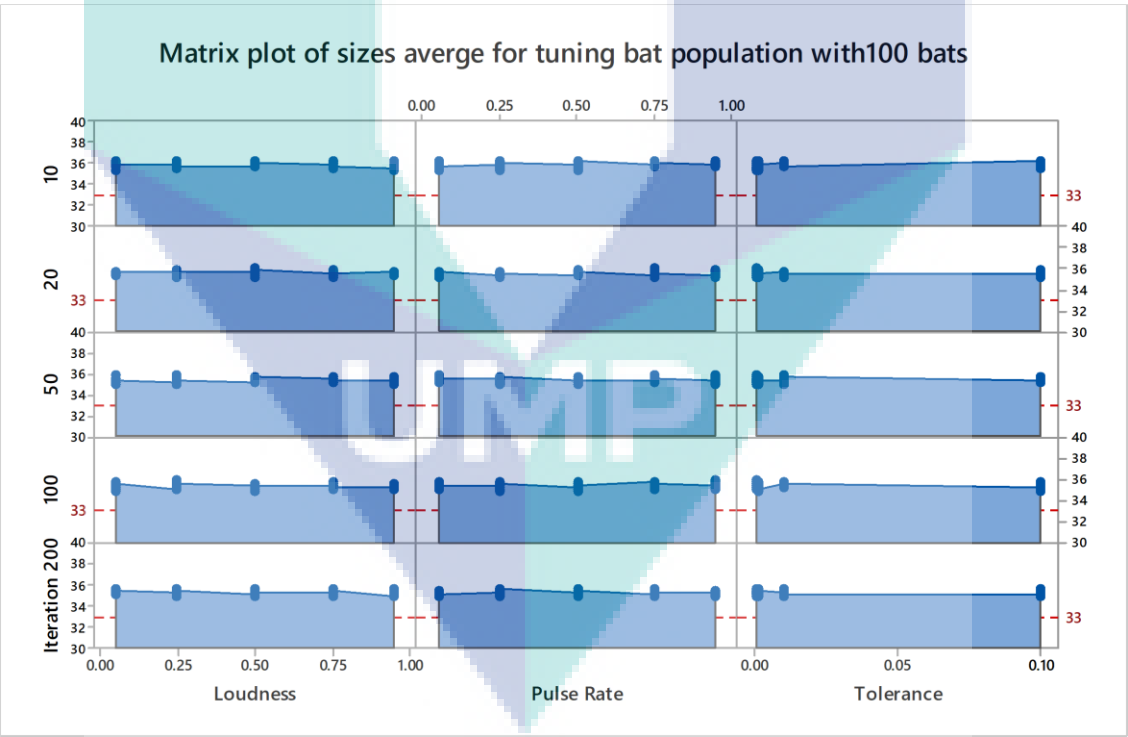


Figure 3.23 The illustration of sizes average with 100 bats.

Based on the empirical experiments and the results of this study shown in Figure 3.16 to Figure 3.23, the differences in the test suite slightly decreased when the bat population and generation increases for values more than 50. The same observation applies for loudness and pulse rate in case of the value 0.50 and tolerance value of 0.001. Here, the solution gives less variation for the higher values (and achieved more reduction in test sizes).

Empirically, the sizes and averages stabilized when n = 50, T_{max} = 200, A_i = 0.25, r_i = 0.05 and tolerance = 0.001. The most stable value for test suite averages = 34.75, which was around 33 test cases in most of the cases which is better minimal test suite size than what is obtained in previous studies (Alsewari & Zamli, 2012a; Stardom, 2001) . Thus, this setting is selected for the current problem been solved.

The parameter values used for the BTS test suite generation benchmarking are as follows: n =50, T_{max} = 200, A_i = 0.25, r_i = 0.05, Q_{min} = 0, Q_{max} = 2, tolerance = 0.001. Here, the frequency range values were defined to the default setting for BA (as the changing of these setting did not affect the process of test suite generation). As observed from the experiments, the quality of the solution stabilized at this setting (BTS consistently achieved minimal test suite sizes).

3.3 Prototype Implementation

The BTS strategy has been developed by using the Java programming language with JDK 1.8. Java has been selected owing to the cross-platform support. In addition, Java also provides rich GUI APIs that facilitates full executional testing platforms in the future. Figure 3.24 summarizes the BTS interface and Figure 3.25 illustrates the functional hierarchy of BTS.

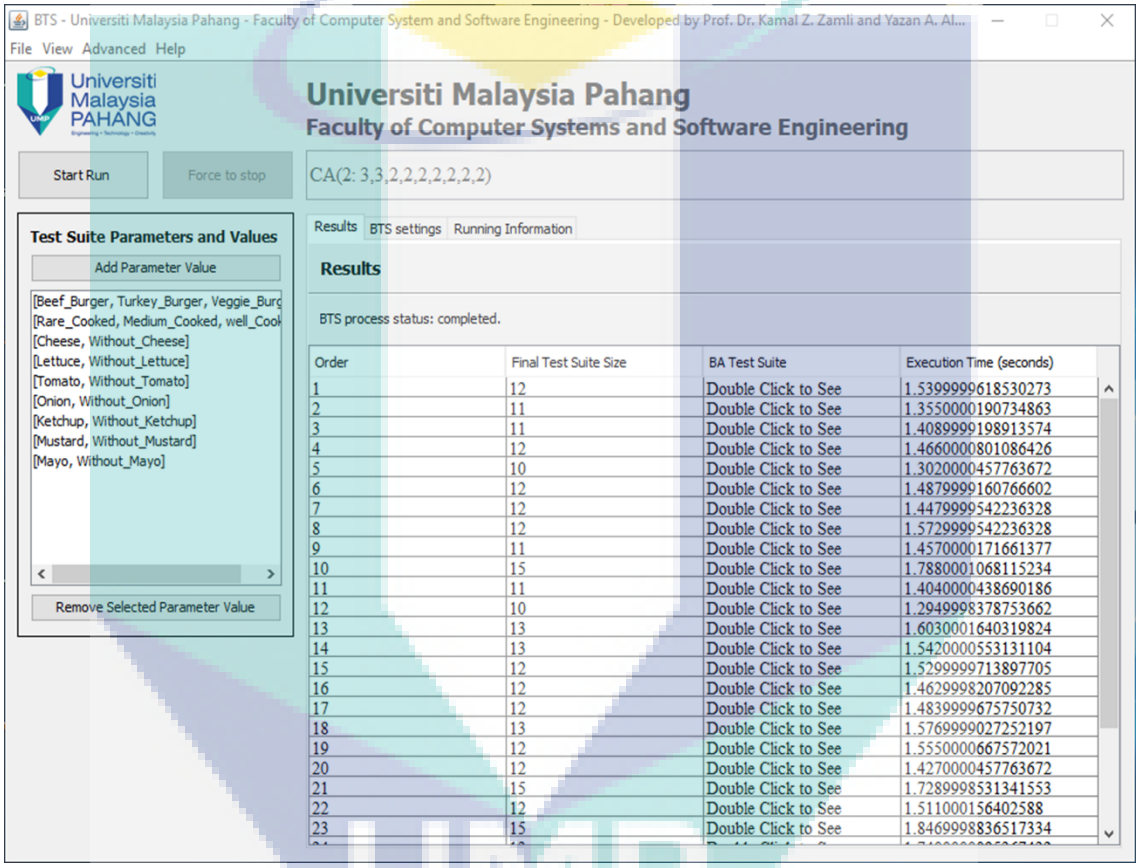


Figure 3.24 The BTS prototype.

The prototype presents the actual data input and the result. Here, a test suite for a pizza selection software is generated based on it actual inputs that represented as $CA(S, 2, 3^2 2^7)$. Here, the optimal test suite obtained is 10 test cases that can be stored into a file or viewed by clicking the test suite record.

The test engineer (i.e. end user) can get FTS result in text file, that can be integrated to a test execution mechanism. Figure 3.25 shows the functional hierarchy for BTS including the main functionally for its four algorithms.

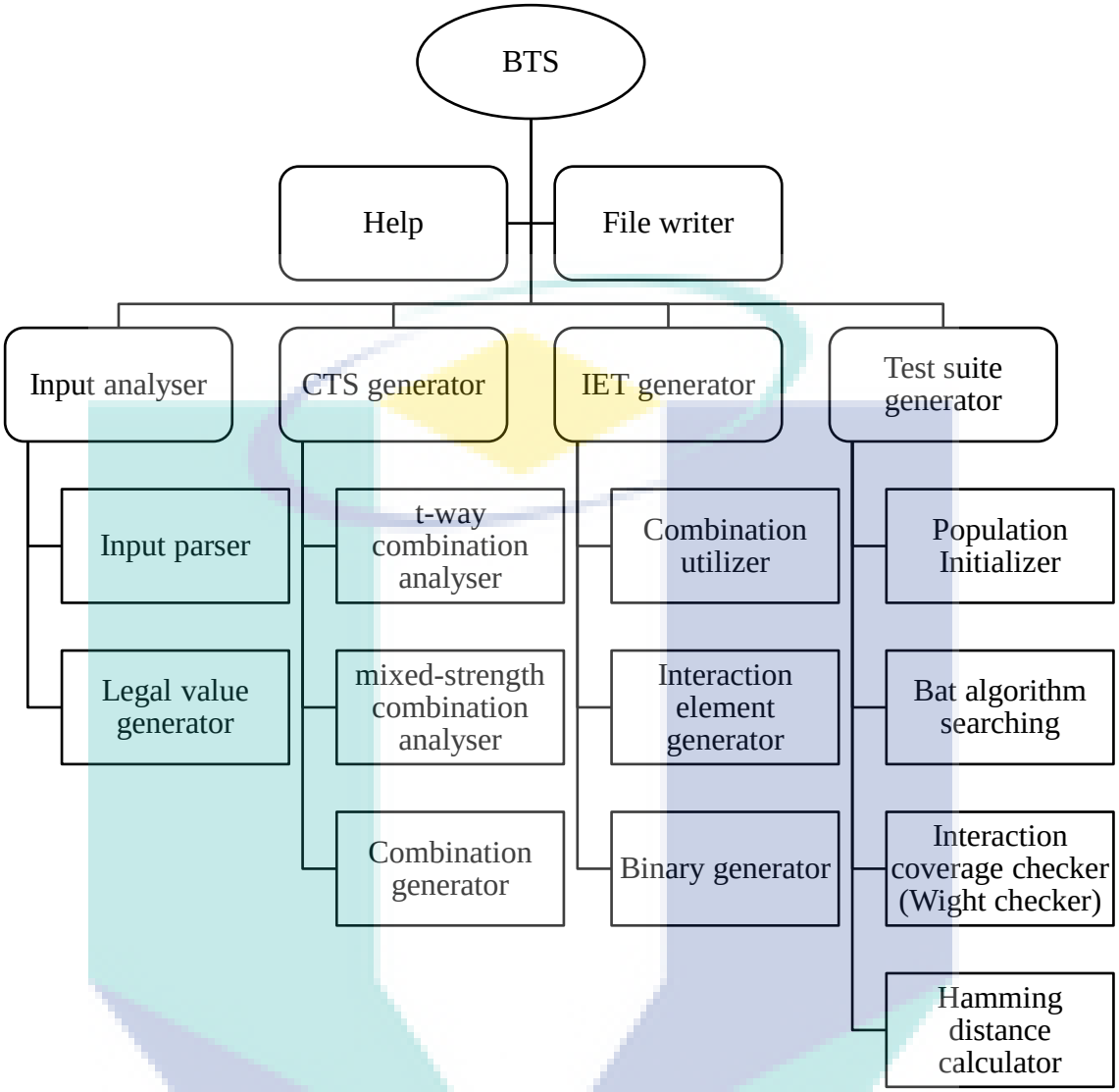


Figure 3.25 The functional hierarchy of BTS.

3.4 Summary

This chapter has provided the full details of the research design concepts of BTS and its algorithms are presented. Additionally, a step-by-step execution of the BTS strategy was presented in this chapter. Furthermore, the tuning and optimizing of the variables used for the BTS test suite generation was presented. Finally, the implementation of the BTS strategy is shown.

The next chapter will present the reports of the BTS benchmarking experiments as well as the statistical analysis based on several real-world applications.

CHAPTER 4

RESULTS AND DISCUSSION

In the previous chapter, the BTS design and implementation were illustrated and elaborated. Furthermore, the BTS parameter settings were optimized based on an elaborate tuning process.

This chapter presents the evaluation of BTS starting with the experimental evaluations. The evaluations consist of the following; the characterization of the original BA and the modified Hamming BA used in the BTS; the comparative benchmarking experiments with well-known t-way strategies that are presented along with the necessary statistical analysis. Finally, the chapter concludes with the summary of the major findings.

4.1 Experimental Evaluations

The experimental evaluation of BTS is focusing on two main goals:

- to characterize the performance of BA against the implemented Hamming BA.
- to benchmark the Hamming BA against other competing approaches.

Based on the aforementioned goals, the BTS evaluation is divided into three parts. In the first part, the size performance and the average time are reported for the BTS variants (the original BA and the Hamming BA). The second part covers the benchmarking of the Hamming BA against its counterparts along with the complete distribution pattern. The best tuning values with a maximum number of iteration (200) and population (50 bats) is adopted (refer to Chapter 3).

The benchmarking of BTS against the other strategies is divided into two sub-sections. The first sub-section reports the results of BTS against other strategies for test suite generation. This is based on previous studies published by strategies' publications (Ahmed et al., 2015; Ahmed et al., 2014; Ahmed & Zamli, 2011b; Ahmed et al., 2012a; Cohen, 2004; Lei et al., 2007; Shiba et al., 2004; Wang & He, 2013; Wang et al., 2008). The second sub-section reports the results of BTS against mixed strength supported strategies. This is also based on previous studies published by strategies' publications (Ahmed & Zamli, 2011b; Ahmed et al., 2012a; Bansal et al., 2015; Cohen et al., 2003c; Wang & He, 2013; Wang et al., 2008; Xiang et al., 2009).

As the strategies in the above-highlighted publications are not publicly available, it is not possible to do the time performance comparison. At best, only the size performances are considered since the size performances are not affected by computing environments (i.e., as currant computer more powerful than the one used in the previous strategies)

The experimental platform comprises of a PC running Windows 7, Intel i7-3770 Quad Core 3.4 GHz CPU, 4 GB RAM, and Java running environments (JRE) version 1.8. All the obtained experimental results are compared and presented in tables together with the benchmarked results from the other strategies' publications. The cells marked "N/A" (not available) indicates that the specific configurations result is not available. Likewise, cells marked "N/S" (not supported) indicates that the specific configuration interaction strength is not supported by the strategy. The best sizes are marked with bold cells. The minimal average sizes are highlighted using dark cells. Whenever necessary, the minimal execution time is marked using italic cells. For the statistical significance, all the BTS (*Size*) results are based on 20 executions. the average size (*Average*) are reported for BTS.

The statistical analysis based on Friedman (Daniel, 1990; Laerd Statistics, 2017a) and Wilcoxon Rank-Sum (Laerd Statistics, 2017b; Wilcoxon, 1945) will be conducted. This is to determine the significance of the results of the undertaken work. The rationale for adopting the Friedman and Wilcoxon Rank-Sum stemmed from the fact that the obtained results are not normally distributed. This presented the need for a non-parametric test. As the benchmarking experiments consider 1 x N comparison, there could be potential significant field-wise errors which can disrupt the statistical conclusions. For this reason, the *post-hoc* analysis using the Bonferonni's Holm

procedure is chosen to adjust the acceptance probability. Then, the obtained results are illustrated for the comparative benchmarking using interval plot. The benchmarking is based on the descriptive means values with the individual standard deviation interval for each of the strategies.

Basically, the null hypothesis (H_0) for the Friedman test is that there is no significant difference between the terms of the test suite sizes median for the results sample at 95 % confident level. Alternatively, the alternative hypothesis (H_1) is that there is a significant difference in terms of the test suite sizes median. This means that the results median distribution is not equal (*less* or *greater*) for the sample. As Friedman test gives a general observation for all the results, a post-hoc test is needed to compare BTS results with the results of each other strategy. As highlighted earlier, the Wilcoxon Rank-Sum test is adopted as the post-hoc method.

The null hypothesis (H_0) for Wilcoxon Rank-Sum test is that there is a significant median difference between the mean pair of samples. The results are compared to other strategies at a 95 % level of confidence level. Here, if the Wilcoxon statistic is less or equal to the alpha ($\alpha = 0.05$) with Bonferroni-holm correction, H_0 will be rejected. Alternatively, H_1 will be adopted if there is a significant difference in terms of the test suite sizes median of BTS with the other strategies. The Bonferroni-holm correction (multiple-comparison correction) is used when several dependent or independent statistical tests are being performed simultaneously. To avoid many spurious positives, the alpha value needs to be lowered to account for the number of comparisons being performed.

The Bonferroni-holm correction value is calculated based on the given alpha for the entire set divided by the number of comparison (m). This gives a critical value with Bonferroni-holm correction α_{holm} for the tested pairs. The Bonferroni-holm correction can be calculated using Equation 4-1.

$$\alpha_{holm} = \frac{\alpha}{m + 1 - k} \tag{4.1}$$

To perform the statistical calculations, the SPSS Statistics Software Version 22 and MiniTab 17 are used. MiniTab is used for interval plotting since SPSS cannot support the interval plot features.

4.2 Experimental Results

In this sub-section, the BTS results are reported in three parts. In the first part, the BTS variants (the original BA and the Hamming BA) size performance and the average time results are reported. The second part reported the benchmarking of the Hamming BA of BTS against its *t*-way counterparts. Then, in the third part, the Hamming BA of BTS is benchmarked against its mixed strength counterparts. The benchmarking in this section is done along with the complete distribution pattern for both *t*-way and mixed strength interaction.

4.2.1 Characterizing BTS

This section highlights the performance of BTS with Hamming BA against the original BA. The configuration consists of a covering array $mCA\ (S, t, 3^7, \{\text{MC}\}), \text{MC} = CA\ (S, 3, 3^4)$ where *t* is varied from $(2 \leq t \leq 6)$. A total of 5 experiments is defined for characterizing BTS as follows:

- Experiment 1: $mCA\ (S, 2, 3^7, \{\text{MC}\}), \text{MC} = CA\ (S, 3, 3^4)$, .
- Experiment 2: $mCA\ (S, 3, 3^7, \{\text{MC}\}), \text{MC} = CA\ (S, 3, 3^4)$).
- Experiment 3: $mCA\ (S, 4, 3^7, \{\text{MC}\}), \text{MC} = CA\ (S, 3, 3^4)$).
- Experiment 4: $mCA\ (S, 5, 3^7, \{\text{MC}\}), \text{MC} = CA\ (S, 3, 3^4)$).
- Experiment 5: $mCA\ (S, 6, 3^7, \{\text{MC}\}), \text{MC} = CA\ (S, 3, 3^4)$).

The obtained results are reported in Table 4.1. The best sizes are marked with bold cells while the minimal average sizes are highlighted using dark cells. The minimal execution time (in seconds) is marked using italic cells.

Table 4.1 The characteristic of BTS (Hamming BA against original BA).

Interaction strength (t)	**BTS with original BA**				**BTS with Hamming BA**			
	best	**worst**	**size**	**time (s)**	**best**	**worst**	**size**	**time (s)**
	Size	**Size**	**Average**	**Average**	**Size**	**Size**	**Average**	**Average**
2	31	37	32.850	*2.652*	**30**	34	32.350	3.678
3	**49**	53	51.700	*13.650*	**49**	53	51.250	14.160
4	155	160	157.150	*91.350*	**153**	159	156.000	97.690
5	**434**	446	440.950	*285.900*	**434**	445	438.050	301.80
6	963	988	976.600	*304.900*	**860**	955	933.400	343.80

The convergence pattern in Figure 4.1 shows the converging of the worst results of the BTS variants; Original BA and Hamming BA.

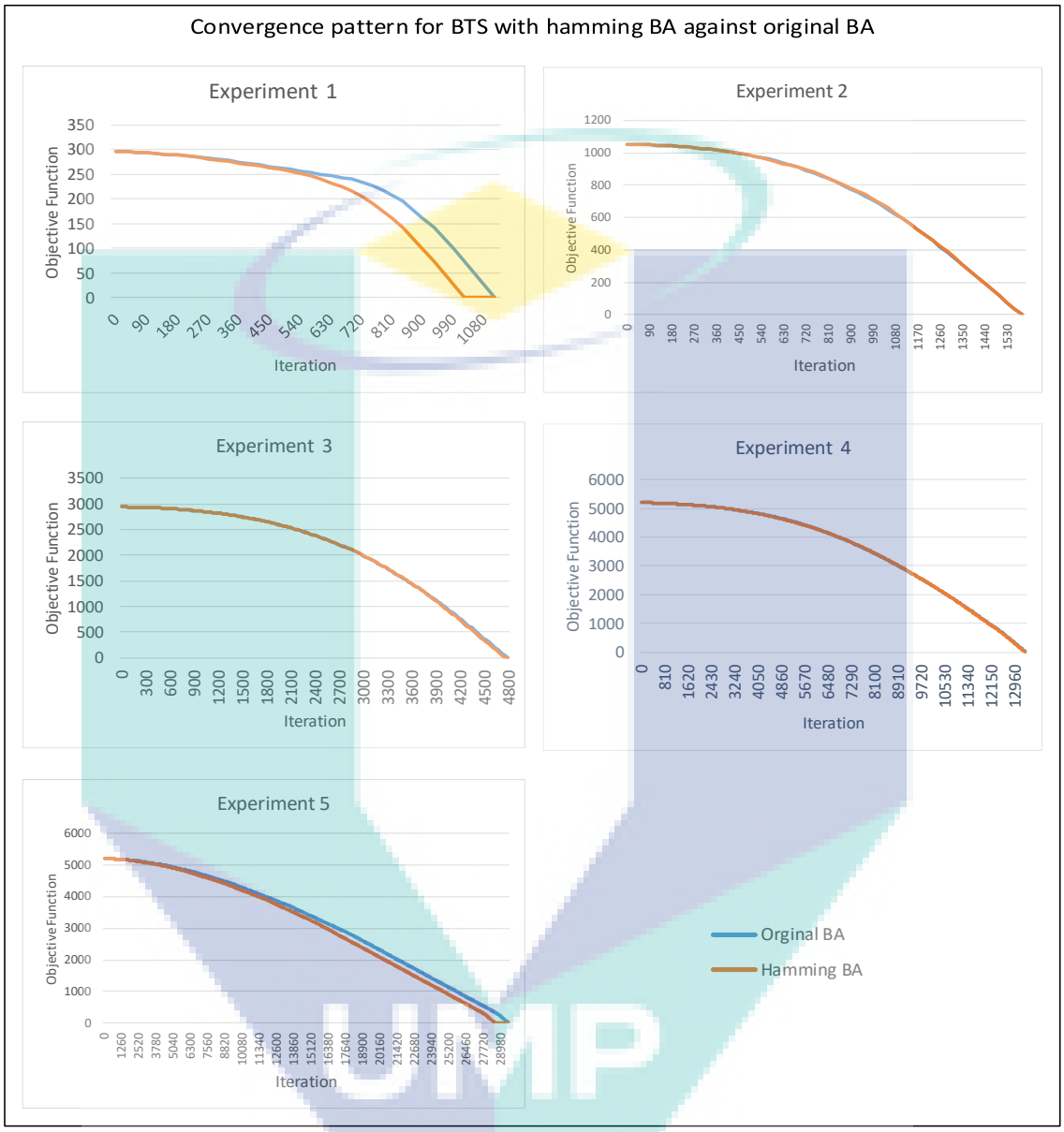


Figure 4.1 The convergence pattern.

In Figure 4.1 the convergence pattern of the two variants of BTS. The Hamming BA variants shows a faster convergence to the optimal test suite with less iterations, while the original variants that uses the default BA gives less time execution. Notably, Hamming BA variants perform better in terms of test suite sizes. Therefore, The Hamming BA is selected for benchmarking with the existing strategies (As the minimal test suite is concerned).

4.2.2 Benchmarking with Other Strategies

The BTS supports interaction strength up to six ($2 \leq t \leq 6$). This is the ideal interaction strength for t-way testing as suggested by (Czarnecki et al., 2012; Kuhn et al., 2010; Kuhn et al., 2015). Therefore, four experimental benchmarking sets that have the specified interaction strength up to t = 6 are conducted. The selected benchmarking test configurations are publicly available in the literature. In the first three experimental sets (experimental sets 1, 2 and 3), the BTS is benchmarked against TConfig, IPOG, ITCH, Jenny, PICT, TVG, PSTG, CS and HSS. In the experimental set 4, the BTS is compared with the results obtained from the execution of PICT, TVG and TConfig. The selected sets of benchmarking experiments are as follows:

- Experimental set 1: The benchmarking for test configurations with varying interaction strengths ($2 \leq t \leq 6$)-way with fixed parameters ($3 \leq P \leq 12$) and ($v = 3$) each (see Table 4.2).
- Experimental set 2: The benchmarking for test configurations with varying interaction strength ($2 \leq t \leq 6$)-way with fixed parameters ($P = 7$) and ($2 \leq v \leq 5$) each (see Table 4.3).
- Experimental set 3: The benchmarking for test configurations with 4-way with interaction strength with varying parameters ($5 \leq P \leq 10$) and fixed ($v = 5$) each (see Table 4.4).
- Experimental set 4: The benchmarking for four real-world software test configurations ($CA(S, t, 2^1 3^5)$, $CA(S, t, 2^1 3^2 5^2)$, $CA(S, t, 2^1 3^3 4^2 5^1 6^1 8^1)$, $CA(S, t, 2^6 3^2 5^1)$) with varying interaction strength ($2 \leq t \leq 6$)-way (see Table 4.5).

Table 4.2 The minimum test suite sizes for experimental set 1.

t-values	*P-values*	**Deterministic**				**Probabilistic**						
		TConfig	**IPOG**	**ITCH**	**Jenny**	**PICT**	**TVG**	**PSTG**	**CS**	**HSS**	**BTS**	
		Size	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Average**
2	3	10	11	9	9	10	10	9	9	9	9	9.7500
	4	10	12	9	13	13	12	9	9	9	9	9.0000
	5	14	14	15	14	13	13	12	11	12	11	11.100
	6	15	15	15	15	14	15	13	13	13	14	14.300
	7	15	17	15	16	16	15	15	14	15	15	15.100
	8	17	17	15	17	16	15	15	15	15	15	15.600
	9	17	17	15	18	17	15	17	16	17	16	16.300
	10	17	20	15	19	18	16	17	17	17	16	16.700
	11	20	20	15	17	18	16	17	18	17	17	17.350
	12	20	20	15	19	19	16	18	18	18	17	17.650
3	4	32	39	27	34	34	34	27	28	30	27	30.100
	5	40	43	45	40	43	41	39	38	39	39	41.050
	6	48	53	45	51	48	49	45	43	45	33	38.300
	7	55	57	45	51	51	55	50	48	50	49	50.750
	8	58	63	45	58	59	60	54	53	54	52	53.150
	9	64	65	75	62	63	64	58	58	59	55	57.300
	10	68	68	75	65	65	68	62	62	62	59	60.750
	11	72	76	75	65	70	69	64	66	66	61	63.600
	12	77	76	75	68	72	70	67	70	67	65	65.950

t-values	*P-values*	**Deterministic**				**Probabilistic**						
		TConfig	**IPOG**	**ITCH**	**Jenny**	**PICT**	**TVG**	**PSTG**	**CS**	**HSS**	**BTS**	
		Size	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Average**
4	5	97	115	153	109	100	105	96	94	94	81	84.200
	6	141	181	153	140	142	139	133	132	132	130	134.40
	7	166	185	216	169	168	172	155	154	154	149	154.20
	8	190	203	216	187	189	192	175	173	174	172	174.55
	9	213	238	306	206	211	215	195	195	195	157	186.15
	10	235	241	336	221	231	233	210	211	212	205	207.25
	11	258	272	348	236	249	250	222	299	223	220	221.25
	12	272	275	372	252	269	268	244	253	244	235	437.60
5	6	305	393	NS	348	310	321	312	304	310	256	279.90
	7	477	608	N/S	458	452	462	441	434	436	434	438.00
	8	583	634	N/S	548	555	562	515	515	515	514	517.10
	9	684	771	N/S	633	637	660	598	590	597	587	592.40
	10	773	784	N/S	714	735	750	667	682	670	659	663.50
	11	858	980	N/S	791	822	833	747	778	753	736	738.75
	12	938	980	N/S	850	900	824	809	880	809	797	848.60
6	7	921	1281	N/S	1087	1015	1024	977	960	977	896	917.80
	8	1515	2098	N/S	1466	1455	1484	1402	1401	1402	1395	1399.5
	9	1931	2160	N/S	1840	1818	1849	1684	1689	1684	1682	1687.4
	10	N/A	2726	N/S	2160	2165	2192	1980	2027	1991	1976	2002.3
	11	N/A	2739	N/S	2459	2496	2533	2255	2298	2255	2192	2237.2
	12	N/A	3649	N/S	2757	2815	2597	2528	2638	2528	2503	2589.4

Table 4.3 The minimum test suite sizes for experimental set 2.

t-values	*v*-values	Deterministic				Probabilistic						
		TConfig	IPOG	ITCH	Jenny	PICT	TVG	PSTG	CS	HSS	BTS	
		Size	Size	Size	Size	Size	Size	Size	Size	Size	Size	Average
2	2	7	8	6	8	7	7	6	6	7	7	7.00000
	3	15	17	15	16	16	15	15	15	14	15	15.0000
	4	28	28	28	28	27	27	26	25	25	24	24.9000
	5	40	42	45	37	40	42	37	37	35	33	35.1000
3	2	16	19	13	14	15	15	13	12	12	15	15.5000
	3	55	57	45	51	51	55	50	49	50	49	50.3500
	4	112	208	112	124	124	134	116	117	121	115	115.900
	5	239	275	225	236	241	260	225	223	223	217	220.200
4	2	36	48	40	31	32	31	29	27	29	31	33.7500
	3	166	185	216	169	168	167	155	155	155	152	154.200
	4	568	509	704	517	529	559	487	487	500	482	485.900
	5	1320	1349	1750	1248	1279	1385	1176	1171	1174	1153	1163.85
5	2	56	128	N/S	57	57	59	53	53	53	54	59.0500
	3	477	608	N/S	458	452	464	441	439	437	435	439.550
	4	1792	2560	N/S	1938	1933	2010	1826	1845	1831	1802	1813.30
	5	N/A	8091	N/S	5895	5814	6257	5474	5479	5468	5417	5430.15
6	2	64	64	N/S	87	72	78	64	66	64	64	64.0000
	3	921	1281	N/S	1087	1015	1016	977	973	916	914	924.550
	4	N/A	4096	N/S	6127	5847	5978	5599	5610	4096	5415	5446.25
	5	N/A	28513	N/S	23492	22502	23218	21595	21597	21748	21436	21371.4

Table 4.4 The minimum test suite sizes for experimental set 3.

P-values	Deterministic				Probabilistic						
	TConfig	IPOG	ITCH	Jenny	PICT	TVG	PSTG	CS	HSS	BTS	
	Size	Size	Size	Size	Size	Size	Size	Size	Size	Size	Average
5	773	908	625	837	810	849	779	776	751	736	741.05
6	1092	1239	625	1074	1072	1128	1001	991	990	965	972.70
7	1320	1349	1750	1248	1279	1384	1209	1200	1186	1158	1162.4
8	1532	1792	1750	1424	1468	1595	1417	1415	1358	1317	1324.6
9	1724	1793	1750	1578	1643	1795	1570	1562	1530	1508	1510.3
10	1878	1965	1750	1719	1812	1917	1716	1731	1624	1746	1763.3

Table 4.5 The minimum test suite sizes for experimental set 4.

t-value	Real-world software	Test configurations	Deterministic	Probabilistic			
			TConfig	PICT	TVG	BTS	
			Size	Size	Size	Size	Average
2	Count	MCA$(S, 2, 2^1 3^5)$	15	14	15	12	12.9000
	Nametbl	$MCA(S, 2, 2^1 3^2 5^2)$	26	25	25	25	25.1500
	Flex	$MCA(S, 2, 2^6 3^2 5^1)$	18	17	19	15	16.7000
	Grep	$MCA(S, 2, 2^1 3^3 4^2 5^1 6^1 8^1)$	53	49	53	48	51.0500
3	Count	$MCA(S, 3, 2^1 3^5)$	44	44	45	33	36.0000
	Nametbl	$MCA(S, 3, 2^1 3^2 5^2)$	82	79	87	75	79.2500
	Flex	$MCA(S, 3, 2^6 3^2 5^1)$	62	53	55	51	54.9000
	Grep	$MCA(S, 3, 2^1 3^3 4^2 5^1 6^1 8^1)$	314	289	291	270	277.950
4	Count	$MCA(S, 4, 2^1 3^5)$	126	113	121	113	116.200
	Nametbl	$MCA(S, 4, 2^1 3^2 5^2)$	248	228	230	225	225.400
	Flex	$MCA(S, 4, 2^6 3^2 5^1)$	149	135	134	134	135.600
	Grep	$MCA(S, 4, 2^1 3^3 4^2 5^1 6^1 8^1)$	1458	1167	1350	1225	1229.90
5	Count	$MCA(S, 5, 2^1 3^5)$	263	251	268	243	245.500
	Nametbl	$MCA(S, 5, 2^1 3^2 5^2)$	450	450	450	450	450.000
	Flex	$MCA(S, 5, 2^6 3^2 5^1)$	349	305	312	291	295.700
	Grep	$MCA(S, 5, 2^1 3^3 4^2 5^1 6^1 8^1)$	5160	4634	5288	4758	4761.00
6	Flex	$MCA(S, 6, 2^6 3^2 5^1)$	732	654	671	612	625.400
	Grep	$MCA(S, 6, 2^1 3^3 4^2 5^1 6^1 8^1)$	14258	15627	17576	13983	14394.3

4.2.3 Benchmarking for Mixed-Strength Test Configurations

In this sub-section, the performance of BTS in terms of test suite sizes for mixed-strength test suite generations is presented. The performance is based on well-known standard benchmark configurations that are publicly available in the related literatures (Ahmed & Zamli, 2011b; Alsewari & Zamli, 2012a; Bansal et al., 2015; Cohen et al., 2003c; Xiang et al., 2009). Four different experimental sets are selected to access the performance of the BTS. The BTS results are compared to the available results of nine well-known t-way strategies that support mixed-strength test suite construction.

The comparative experimental sets are as follows:

- Experimental set 5: The benchmarking results of 18 different sub-strength test configurations with uniform mCA $(S, 2, 3^{15}, \{MC\})$ as main-strength (see Table 4.6).
- Experimental set 6: The benchmarking results of 13 different sub-strength test configurations with mixed-strength mMCA $(S, 2, 4^3\ 5^3\ 6^2, \{MC\})$ as main-strength (see Table 4.7).
- Experimental set 7: The benchmarking results of 11 different sub-strength test configurations with mixed-strength mMCA $(S, 2, 10^1\ 9^1\ 8^1\ 7^1\ 6^1\ 5^1\ 4^1\ 3^1\ 2^1, \{MC\})$ as main-strength (see Table 4.8).
- Experimental set 8: The benchmarking results of 6 different sub-strength test configurations with mixed-strength mMCA $(S, 2, 3^{30}10^2, \{MC\})$ as main-strength (see Table 4.9).

The four comparative mixed-strength experimental sets (Table 4.6 to Table 4.9) show the comparative results of BTS against ITCH (Hartman et al., 2005), IPOG (Lei et al., 2007), TVG (Arshem, 2003), PICT (Czerwonka, 2006), SA-Mayer (Cohen et al., 2003c), ACS-VSITs (Xiang et al., 2009), PWiseGen-VSCA (Bansal et al., 2015), VS-PSTG (Ahmed & Zamli, 2011b) and HSS (Alsewari & Zamli, 2012a).

Table 4.6 The minimum test suite sizes for experimental set 5.

Test configurations	**Deterministic**		**Probabilistic**								
$mCA\ (S, 2, 3^{15}, \{MC\})$	**ITCH**	**IPOG**	**TVG**	**PICT**	**SA-Mayer**	**ACS-VSITs**	**PWiseGen-VSCA**	**VS-PSTG**	**HSS**	**BTS**	
$\{MC\}$	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Average**
∅	31	21	22	35	16	19	16	19	20	19	19.70
$CA\ (S, 3, 3^3)$	48	27	27	81	27	27	27	27	27	27	27.30
$CA\ (S, 3, 3^3)^2$	59	30	30	739	27	27	27	27	27	27	27.70
$CA\ (S, 3, 3^3)^3$	69	33	30	785	27	27	27	27	27	27	28.00
$CA\ (S, 3, 3^4)$	59	39	35	105	27	27	27	30	27	30	32.10
$CA\ (S, 3, 3^5)$	62	39	41	131	33	38	33	38	38	39	40.70
$CA\ (S, 4, 3^4)$	103	81	81	245	N/S	N/S	81	81	81	81	81.00
$CA\ (S, 4, 3^5)$	118	122	103	301	N/S	N/S	91	97	94	90	97.30
$CA\ (S, 4, 3^7)$	189	181	168	505	N/S	N/S	158	158	159	154	155.6
$CA\ (S, 5, 3^5)$	261	243	243	730	N/S	N/S	243	243	243	243	243.0
$CA\ (S, 5, 3^7)$	481	581	462	1356	N/S	N/S	441	441	441	429	438.7
$CA\ (S, 6, 3^6)$	745	729	729	2187	N/S	N/S	729	729	729	729	729.0
$CA\ (S, 6, 3^7)$	1050	1196	1028	3045	N/S	N/S	N/A	966	902	950	963.1
$CA\ (S, 3, 3^4)$, $CA\ (S, 3, 3^5)$, $CA\ (S, 3, 3^6)$	114	51	53	1376	34	40	40	45	45	43	45.50
$CA\ (S, 3, 3^6)$	61	53	48	146	34	45	40	45	45	45	46.40
$CA\ (S, 3, 3^7)$	68	58	54	154	41	48	47	49	51	47	49.70
$CA\ (S, 3, 3^9)$	94	65	62	177	50	57	57	57	62	56	57.60
$CA\ (S, 3, 3^{15})$	132	N/S	81	83	67	76	74	74	77	73	74.70

Table 4.7 The minimum test suite sizes for experimental set 6.

Test configurations	**Deterministic**		**Probabilistic**								
$\boldsymbol{mMCA\ (S, 2, 4^3 5^3 6^2, \{MC\})}$	**ITCH**	**IPOG**	**TVG**	**PICT**	**SA-Mayer**	**ACS-VSITs**	**PWiseGen-VSCA**	**VS-PSTG**	**HSS**	**BTS**	
{MC}	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Average**
∅	48	43	44	43	36	41	37	42	42	41	42.300
$CA(S, 3, 4^3)$	97	83	67	384	64	64	64	64	64	64	64.100
$MCA(S, 3, 4^3 5^2)$	164	147	132	781	100	104	120	124	116	122	125.00
$MCA(S, 3, 5^3)$	145	136	125	750	125	125	125	125	125	125	125.00
$MCA(S, 4, 4^3 5^1)$	354	329	320	1920	N/S	N/S	320	320	320	453	463.70
$MCA(S, 5, 4^3 5^2)$	1639	1602	1600	9600	N/S	N/S	1600	1600	1600	1600	1600.0
$CA(S, 3, 4^3)$, $CA(S, 3, 5^3)$	194	136	125	8000	125	125	125	125	125	125	125.00
$MCA(S, 4, 4^3 5^1)$, $MCA(S, 4, 5^2 6^2)$	1220	900	900	NA	N/S	N/S	900	900	900	900	900.00
$MCA(S, 4, 4^3 5^2)$	510	512	496	2874	N/S	N/S	472	472	453	456	465.70
$MCA(S, 5, 4^3 5^2)$	2520	2763	2592	15048	N/S	N/S	2430	2430	2430	2380	2409.6
$MCA(S, 3, 4^3 5^3 6^1)$	254	215	237	1266	171	201	206	206	212	204	208.60
$MCA(S, 3, 5^1 6^2)$	188	180	180	900	180	180	180	180	180	180	180.00
$MCA(S, 3, 4^3 5^3 6^2)$	312	N/S	302	261	214	255	260	260	263	256	259.50

Table 4.8 The minimum test suite sizes for experimental set 7.

Test configurations	**Deterministic**		**Probabilistic**								
$\boldsymbol{mMCA\ (S, 2, 10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1, \{MC\})}$	**ITCH**	**IPOG**	**TVG**	**PICT**	**SA-Mayer**	**ACS-VSITs**	**PWiseGen-VSCA**	**VS-PSTG**	**HSS**	**BTS**	
{MC}	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Average**
∅	119	91	99	102	N/A	N/A	92	97	94	93	96.500
$MCA\ (S, 3, 10^1 9^1 8^1)$	765	720	720	31256	N/A	N/A	720	720	720	720	720.00
$MCA\ (S, 3, 7^1 6^1 5^1)$	301	221	210	19515	N/A	N/A	210	210	210	210	210.30
$MCA\ (S, 3, 4^1 3^1 2^1)$	140	91	99	2397	N/A	N/A	92	97	94	94	96.000
$MCA\ (S, 3, 10^1 9^1 8^1 7^1)$	806	772	784	22878	N/A	N/A	740	742	740	735	742.10
$MCA\ (S, 3, 10^1 9^1 8^1)$, $MCA\ (S, 3, 7^1 6^1 5^1)$	947	720	720	N/A	N/A	N/A	720	720	720	720	720.00
$MCA\ (S, 3, 10^1 9^1 8^1)$, $MCA\ (S, 6, 7^1 6^1 5^1 4^1 3^1 2^1)$	5803	5041	5040	N/A	N/A	N/A	N/A	5040	5040	5040	5043.2
$MCA\ (S, 3, 10^1 9^1 8^1)$, $MCA\ (S, 3, 7^1 6^1 5^1)$, $MCA\ (S, 3, 4^1 3^1 2^1)$	968	720	720	N/A	N/A	N/A	720	720	720	720	720.00
$MCA\ (S, 4, 5^1 4^1 3^1 2^1)$	237	142	123	1200	N/A	N/A	120	120	120	120	120.00
$MCA\ (S, 5, 10^1 9^1 4^1 3^1 2^1)$	2276	2160	2160	124157	N/A	N/A	2160	2160	2160	2160	2160.0
$MCA\ (S, 6, 7^1 6^1 5^1 4^1 3^1 2^1)$	5157	5041	5040	N/A	N/A	N/A	5040	5040	5040	5040	5040.0

Table 4.9 The minimum test suite sizes for experimental set 8.

Test configurations	**Deterministic**		**Probabilistic**								
$\boldsymbol{mMCA\ (S, 2, 3^{30}10^{2}, \{MC\})}$	**ITCH**	**IPOG**	**TVG**	**PICT**	**SA-Mayer**	**ACS-VSITs**	**PWiseGen-VSCA**	**VS-PSTG**	**HSS**	**BTS**	
{MC}	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Size**	**Average**
∅	N/A	101	101	100	100	100	N/A	102	106	107	114.10
$CA\ (S, 3, 3^{20})$	N/A	100	103	940	100	100	N/A	105	109	105	106.40
$MCA(S, 3, 3^{20}10^{2})$	N/A	N/S	423	423	304	396	N/A	481	450	466	473.20
$MCA(S, 4, 3^{3}10^{1})$	N/A	273	270	810	N/A	N/A	N/A	270	270	270	270.00
$MCA(S, 5, 3^{3}10^{2})$	N/A	2700	2700	2800	N/A	N/A	N/A	2700	2700	2700	2700.0
$MCA(S, 6, 3^{4}10^{2})$	N/A	8100	8100	N/A	N/A	N/A	N/A	8100	8100	8100	8100.0

4.3 Statistical Analysis of the Experimental Results

The statistical analysis is performed using the Friedman and Wilcoxon signed-rank test with Bonferroni-holm correction (α_{holm}) at 95 % confident level (i.e. $\alpha = 0.05$). Additionally, an interval plot of the obtained results (the mean) by each compared strategies is drawn. The interval plots depict the obtained result distributions and their means at a 95 % confidence level (or *confidence interval* (CI))

In this section, the statistical analysis is divided into two sub-sections. The first sub-section considers the results of *t*-way benchmarking while the second sub-section considers the results of the mixed-strength benchmarking. The strategies with N/A and N/S results are considered incomplete and ignored samples as there is no available result for the specified test configuration.

4.3.1 Statistical Analysis for *t*-way Results

The statistical analysis is reported in Tables 4.10 to 4.17. The four interval plots for the mean distributions of individual strategy results (test suite sizes means) are also shown in the aforementioned tables. Figures 4.1 to 4.4 shows each strategy descriptive results' distribution and the resulting mean values.

Table 4.10 Friedman test for Table 5.2.

Friedman	**Conclusion**
Degree of freedom = 7, α =0.05 Friedman statistic (p-value) = 8.310E-42 Chi-square value (X^2) = 210.100	8.310E-42 < 0.05 (i.e. p-value < α). Thus, reject H_0 and proceed to the post-hoc test.

Note: the results for (TConfig and ITCH) are ignored.

Table 4.11 Wilcoxon signed-rank (Post-hoc) test for Table 5.2.

Pair comparison	**p-value**	**α_{holm}**	**Conclusion**
BTS vs IPOG	0.0000000354	0.00714286	Reject H_0
BTS vs PICT	0.0000000517	0.00833333	Reject H_0
BTS vs Jenny	0.0000000520	0.01000000	Reject H_0
BTS vs TVG	0.0000002934	0.01250000	Reject H_0
BTS vs HSS	0.0000003324	0.01666667	Reject H_0
BTS vs PSTG	0.0000004984	0.02500000	Reject H_0
BTS vs CS	0.0000017114	0.05000000	Reject H_0

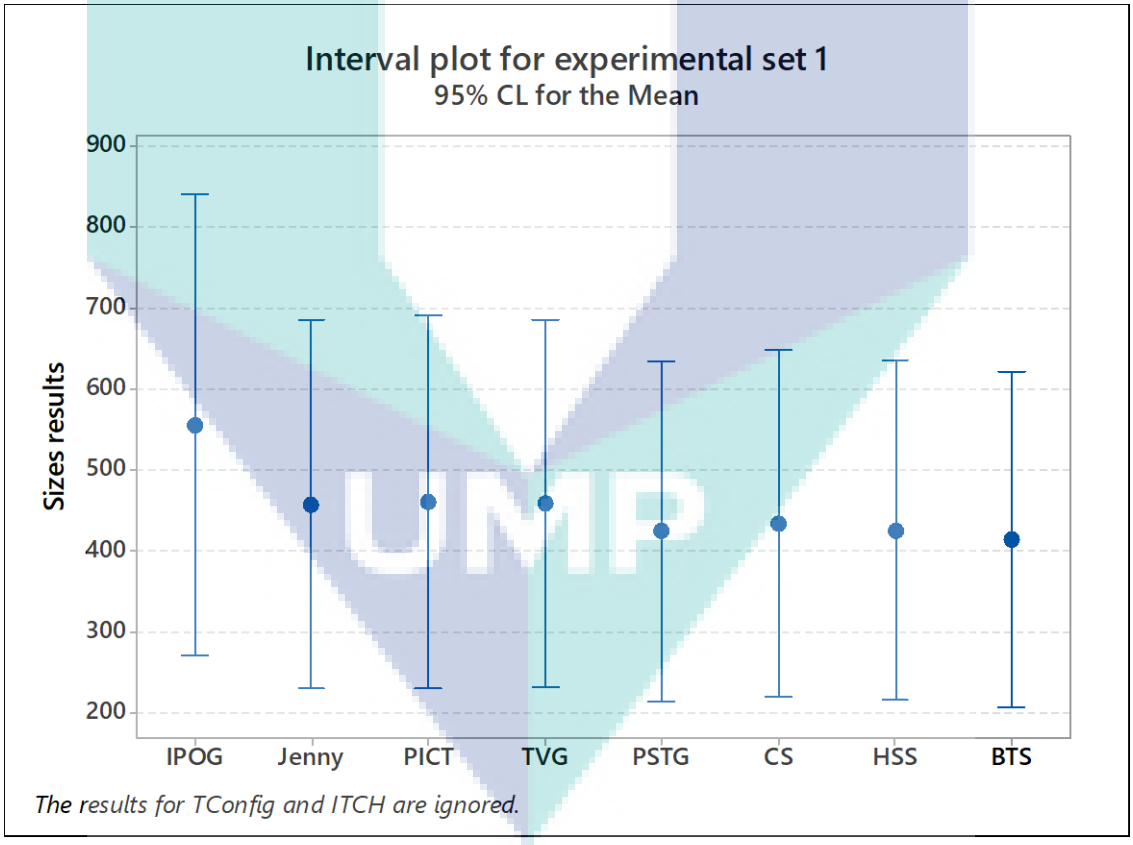


Figure 4.2 The illustration of Table 5.2 results' intervals with CL 95%.

Table 4.12 Friedman test for Table 5.3.

Friedman	**Conclusion**
Degree of freedom = 7, α =0.05 Friedman statistic (p-value) = 6.8673E-18 Chi-square value (X^2) = 96.104	6.8673E-18 < 0.05 Thus, reject H_0 and proceed to the post-hoc test.

Note: the results for (TConfig and ITCH) are ignored.

Table 4.13 Wilcoxon signed-rank (Post-hoc) test for Table 5.3.

Pair comparison	**p-value**	**α_{holm}**	**Conclusion**
BTS vs PICT	0.0002745713	0.00714286	Reject H_0
BTS vs Jenny	0.0002876189	0.00833333	Reject H_0
BTS vs TVG	0.0004377772	0.01000000	Reject H_0
BTS vs IPOG	0.0016940519	0.01250000	Reject H_0
BTS vs PSTG	0.0052675313	0.01666667	Reject H_0
BTS vs CS	0.0089096180	0.02500000	Reject H_0
BTS vs HSS	0.0701753156	0.05000000	Retain H_0

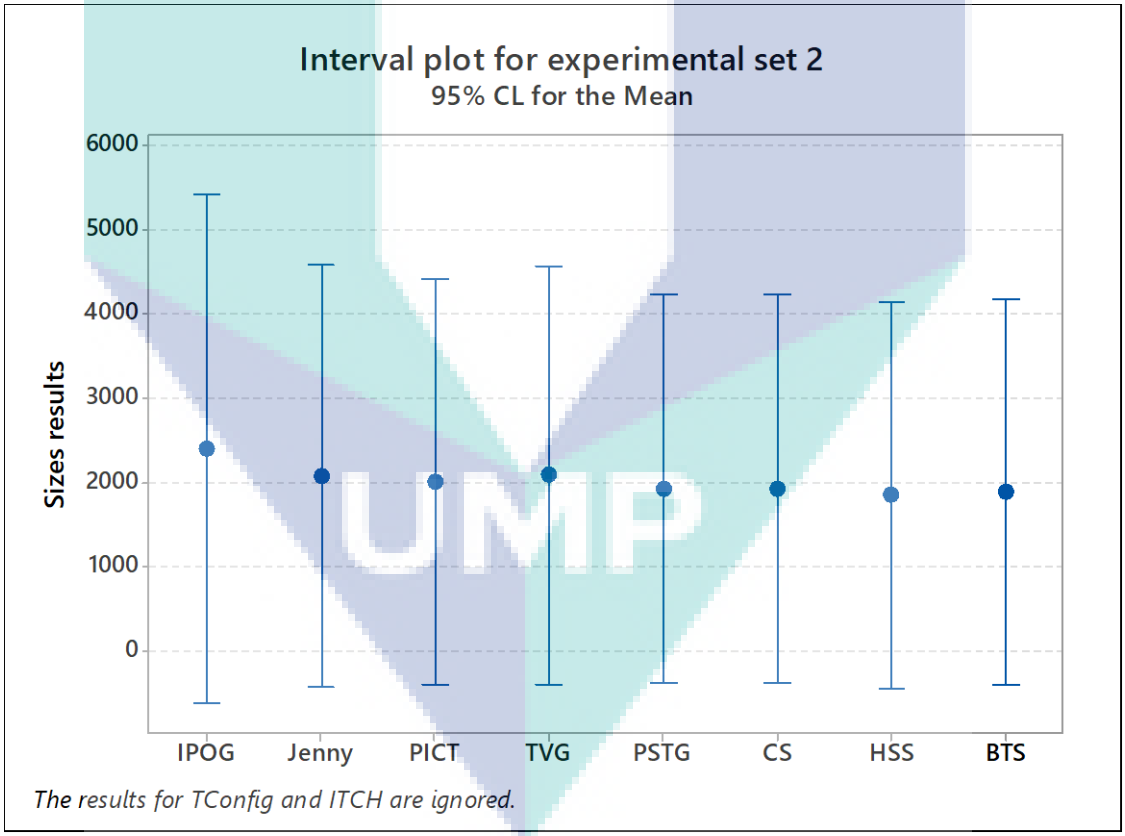


Figure 4.3 The illustration of Table 5.3 results' intervals with CL 95%.

Table 4.14 Friedman test for Table 5.4.

Friedman	**Conclusion**
Degree of freedom = 9, α =0.05 Friedman statistic (p-value) = 1.2330E-5 Chi-square value (X^2) = 38.836	1.2330E-5 < 0.05 Thus, reject H_0 and proceed to the post-hoc test.

Table 4.15 Wilcoxon signed-rank (Post-hoc) test for Table 5.4.

Pair comparison	**p-value**	**α_{holm}**	**Conclusion**
BTS vs TConfig	0.02770785	0.00555556	Retain H_0
BTS vs IPOG	0.02770785	0.00625000	Retain H_0
BTS vs PICT	0.02770785	0.00714286	Retain H_0
BTS vs TVG	0.02770785	0.00833333	Retain H_0
BTS vs Jenny	0.04639946	0.01000000	Retain H_0
BTS vs PSTG	0.04639946	0.01250000	Retain H_0
BTS vs CS	0.04639946	0.01666667	Retain H_0
BTS vs ITCH	0.34544753	0.02500000	Retain H_0
BTS vs HSS	0.34544753	0.05000000	Retain H_0

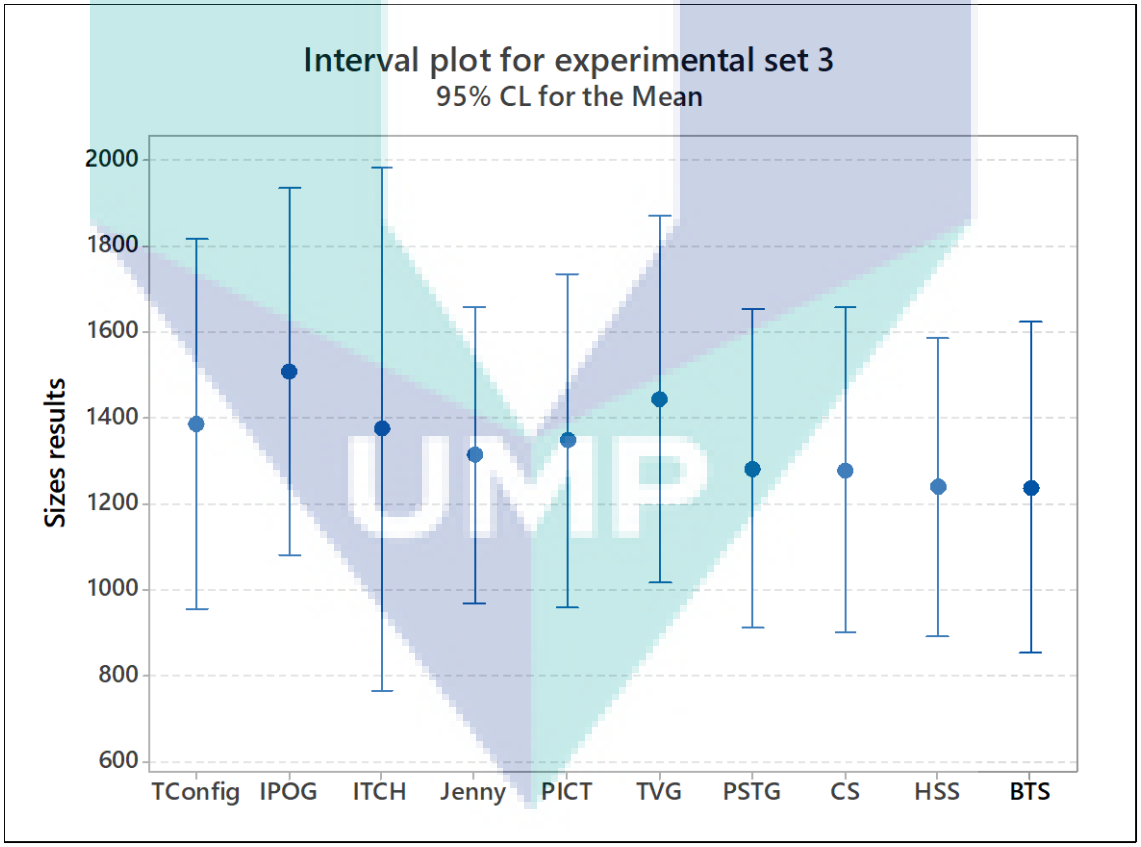


Figure 4.4 The illustration of Table 5.4 results' intervals with CL 95%.

Table 4.16 Friedman test for Table 5.5.

Friedman	**Conclusion**
Degree of freedom = 3, α =0.05 Friedman statistic (p-value) = 4.6293E-8 Chi-square value (X^2) = 36.988	4.6293E-8 < 0.05 Thus, reject H_0 and proceed to the post-hoc test.

Table 4.17 Wilcoxon signed-rank (Post-hoc) test for Table 5.5.

Pair comparison	**p-value**	**α_{holm}**	**Conclusion**
BTS vs TConfig	0.00029190	0.01666667	Reject H_0
BTS vs TVG	0.00064839	0.02500000	Reject H_0
BTS vs PICT	0.06063245	0.05000000	Retain H_0

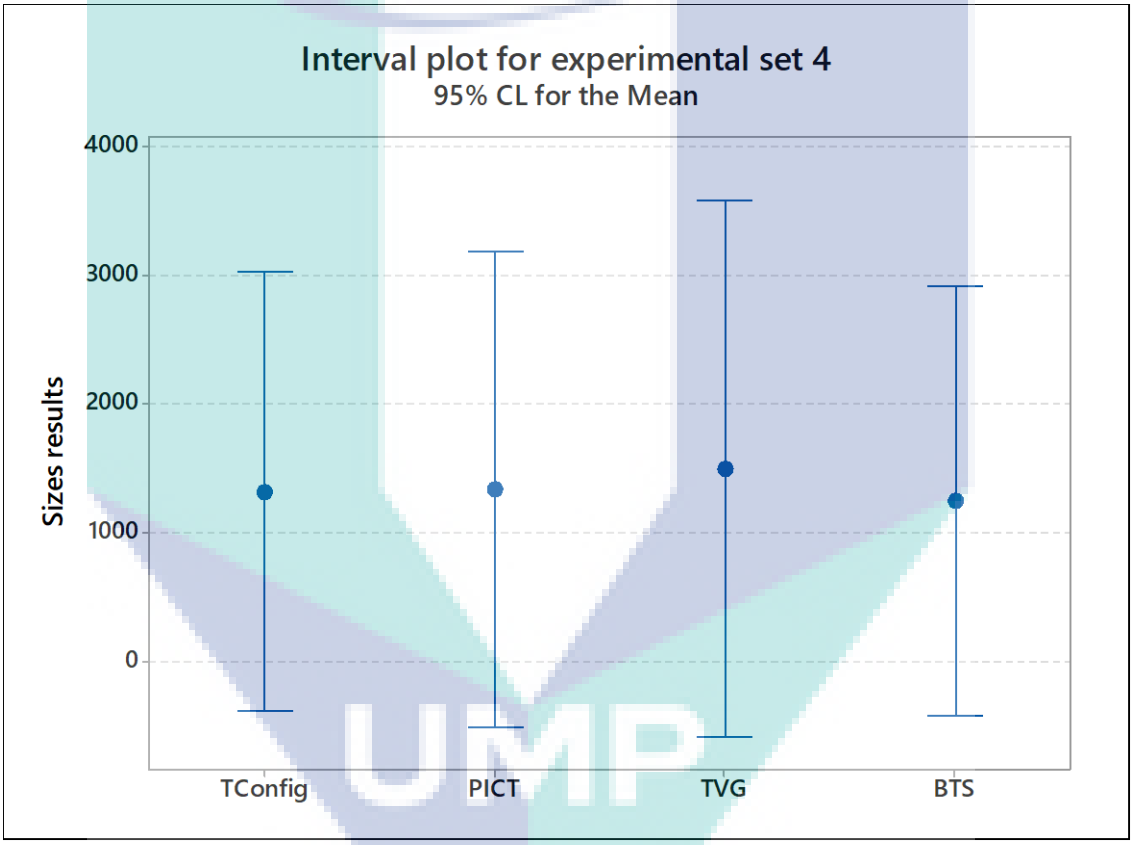


Figure 4.5 The illustration of Table 5.5 results' intervals with CL 95%.

4.3.2 Statistical Analysis of Mixed-Strength Results

The statistical analysis of the mixed-strength benchmarking is reported in Tables 4.18 to 4.24. Also reported in the tables along with the four interval plots of the mean size distribution of each individual strategy results (test suite sizes). A general observation test (Friedman test) is reported first; then, in case of a statistical significant,

a post-hoc test is performed. Figures 4.6 to 4.9 show each strategy's descriptive result distributions and the mean value.

Table 4.18 Friedman test for Table 5.6.

Friedman	**Conclusion**
Degree of freedom = 5, α =0.05 Friedman statistic (p-value) = 5.1268E-16 Chi-square value (X^2) = 81.023	5.1268E-16 < 0.05 Thus, reject H_0 and proceed to the post-hoc test.

Note: the results for (IPOG, SA-Mayer, ACS-VSITs, and PWiseGen-VSCA) are ignored.

Table 4.19 Wilcoxon signed-rank (Post-hoc) test for Table 5.6.

Pair comparison	**p-value**	**α_{holm}**	**Conclusion**
BTS vs ITCH	0.00019575	0.01000000	Reject H_0
BTS vs PICT	0.00019644	0.01250000	Reject H_0
BTS vs TVG	0.00095345	0.01666667	Reject H_0
BTS vs PSTG	0.01471359	0.02500000	Reject H_0
BTS vs HSS	0.14138133	0.05000000	Retain H_0

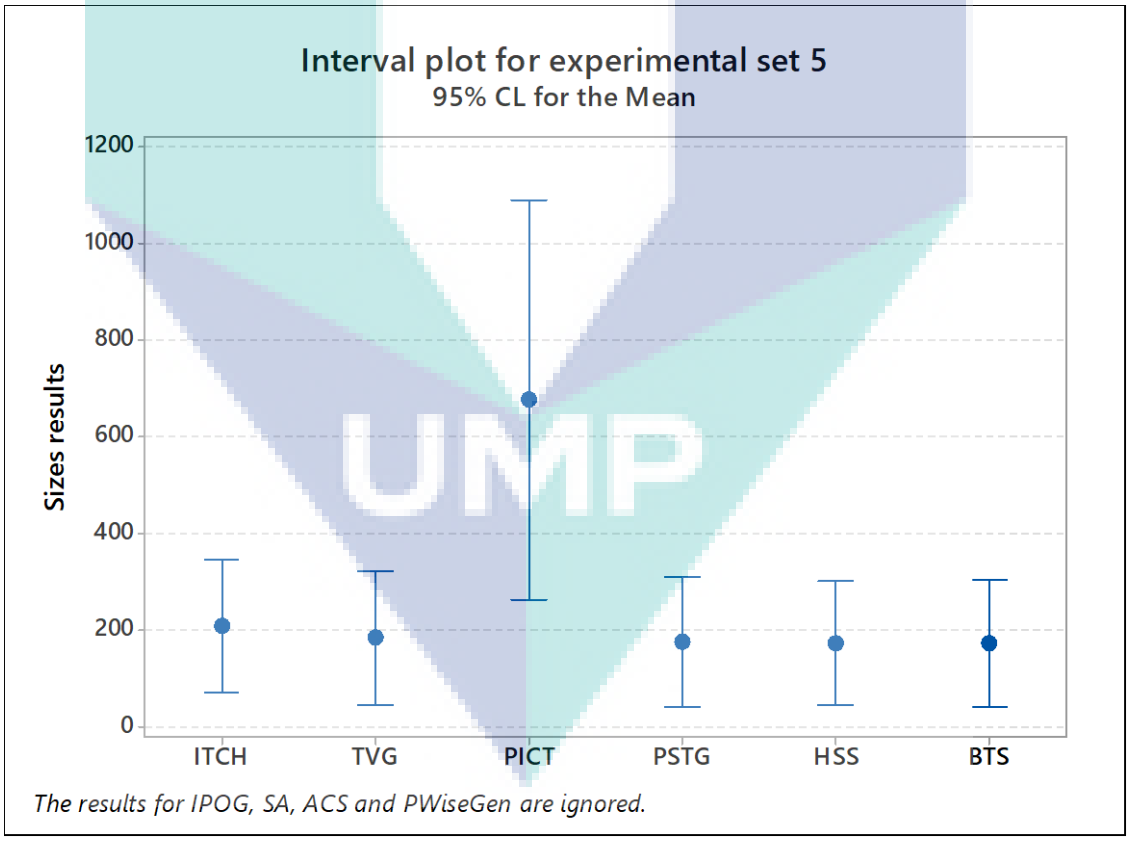


Figure 4.6 The illustration of Table 5.6 results' intervals with CL 95%.

Table 4.20 Friedman test for Table 5.7.

Friedman	**Conclusion**
Degree of freedom = 5, α =0.05 Friedman statistic (p-value) = 5.1739E-8 Chi-square value (X^2) = 42.278	5.1739E-8 < 0.05 Thus, reject H_0 and proceed to the post-hoc test.

Note: the results for (IPOG, PICT, SA-Mayer and ACS-VSITs) are ignored.

Table 4.21 Wilcoxon signed-rank (Post-hoc) test for Table 5.7.

Pair comparison	**p-value**	**α_{holm}**	**Conclusion**
BTS vs ITCH	0.01590644	0.01000000	Retain H_0
BTS vs TVG	0.123025194	0.01250000	Retain H_0
BTS vs PSTG	0.235885211	0.01666667	Retain H_0
BTS vs PWiseGen-VSCA	0.734402143	0.02500000	Retain H_0
BTS vs HSS	0.735316691	0.05000000	Retain H_0

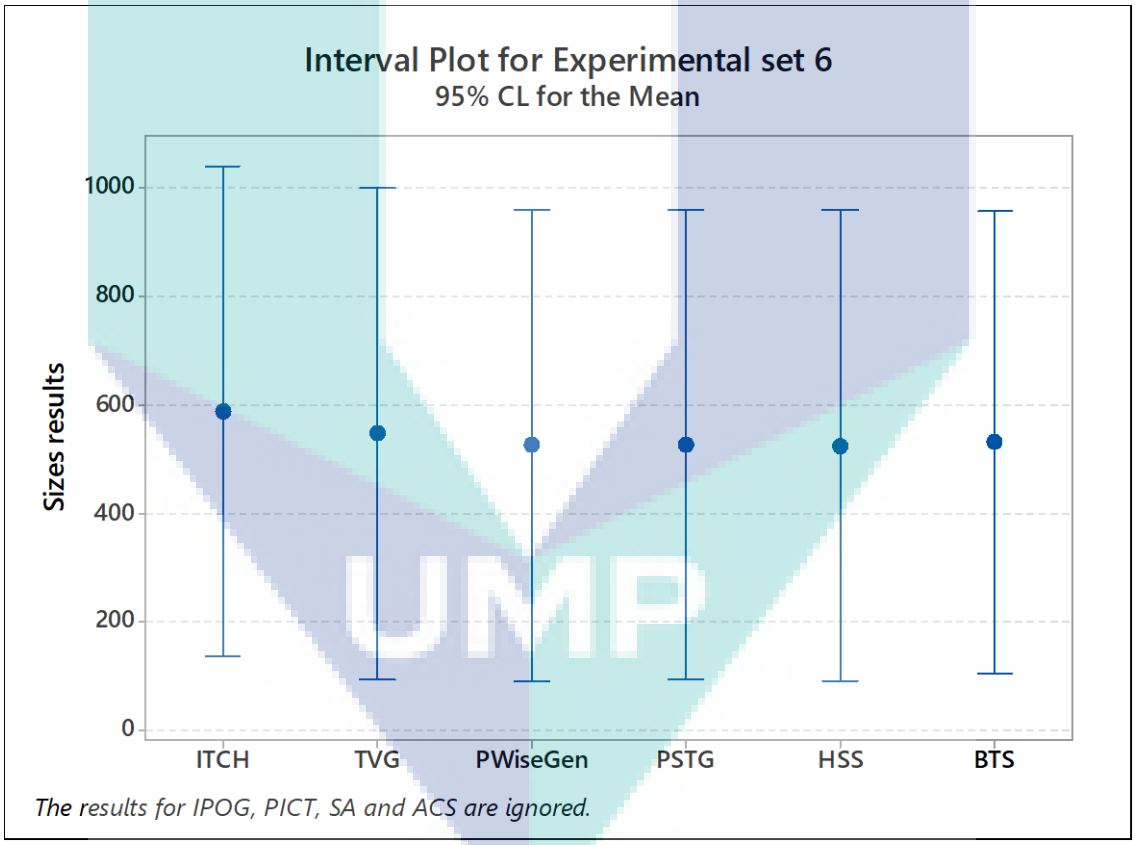


Figure 4.7 The illustration of Table 5.7 results' intervals with CL 95%.

Table 4.22 Friedman test for Table 5.8.

Friedman	**Conclusion**
Degree of freedom = 5, α =0.05 Friedman statistic (p-value) = 4.5553E-7 Chi-square value (X^2) = 37.593	4.5553E-7 < 0.05 Thus, reject H_0 and proceed to the post-hoc test.

Note: the results for (PICT, SA-Mayer, ACS-VSITs and PWiseGen-VSCA) are ignored.

Table 4.23 Wilcoxon signed-rank (Post-hoc) test for Table 5.8.

Pair comparison	**p-value**	**α_{holm}**	**Conclusion**
BTS vs ITCH	0.00333001	0.01000000	Reject H_0
BTS vs TVG	0.067889155	0.01250000	Retain H_0
BTS vs PSTG	0.10880943	0.01666667	Retain H_0
BTS vs HSS	0.179712495	0.02500000	Retain H_0
BTS vs IPOG	0.235885211	0.05000000	Retain H_0

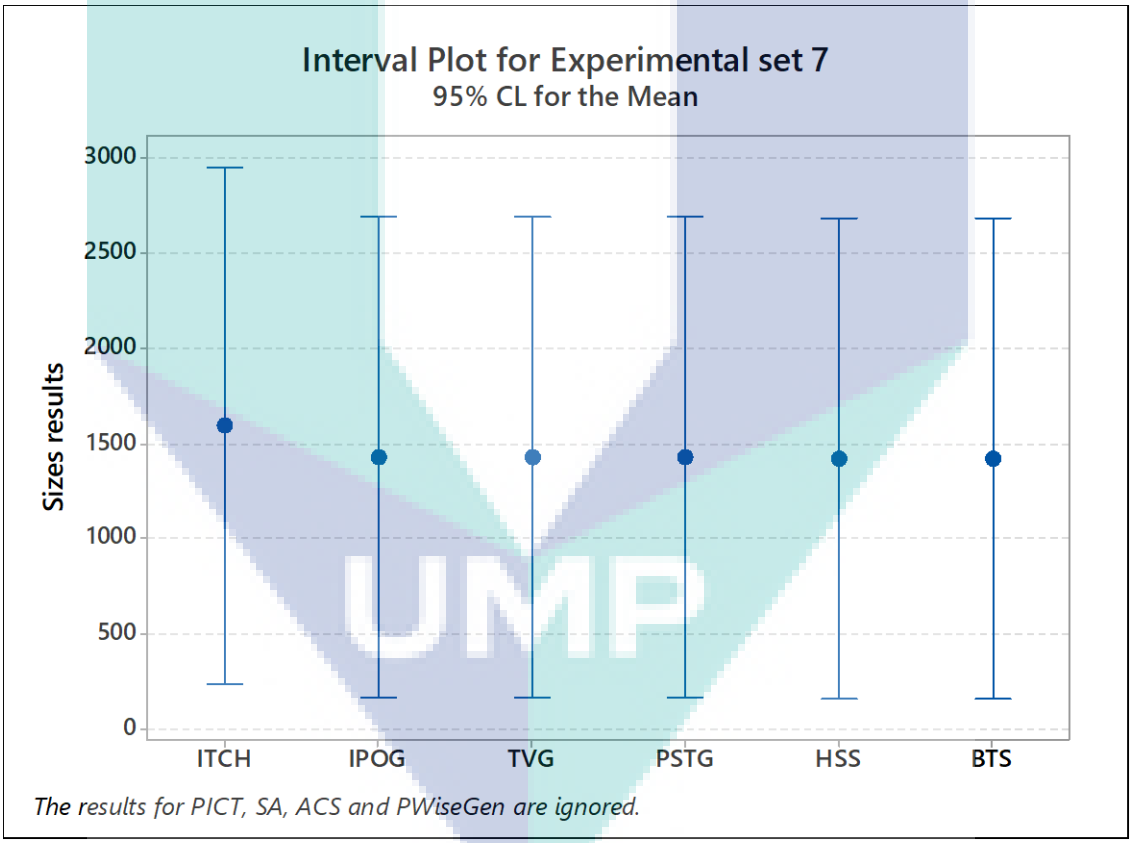


Figure 4.8 The illustration of Table 5.8 results' intervals with CL 95%.

Table 4.24 Friedman test for Table 5.9.

Friedman	**Conclusion**
Degree of freedom = 3, α =0.05 Friedman statistic (p-value) = 0.1277 Chi-square value (X^2) = 5.690	4.5553E-7 < 0.05 Thus, retain H_0, there no statistical significant (the post-hoc test is not required in this case).

Note: the results for (ITCH, IPOG, PICT, SA-Mayer, ACS-VSITs and PWiseGen-VSCA) are ignored.

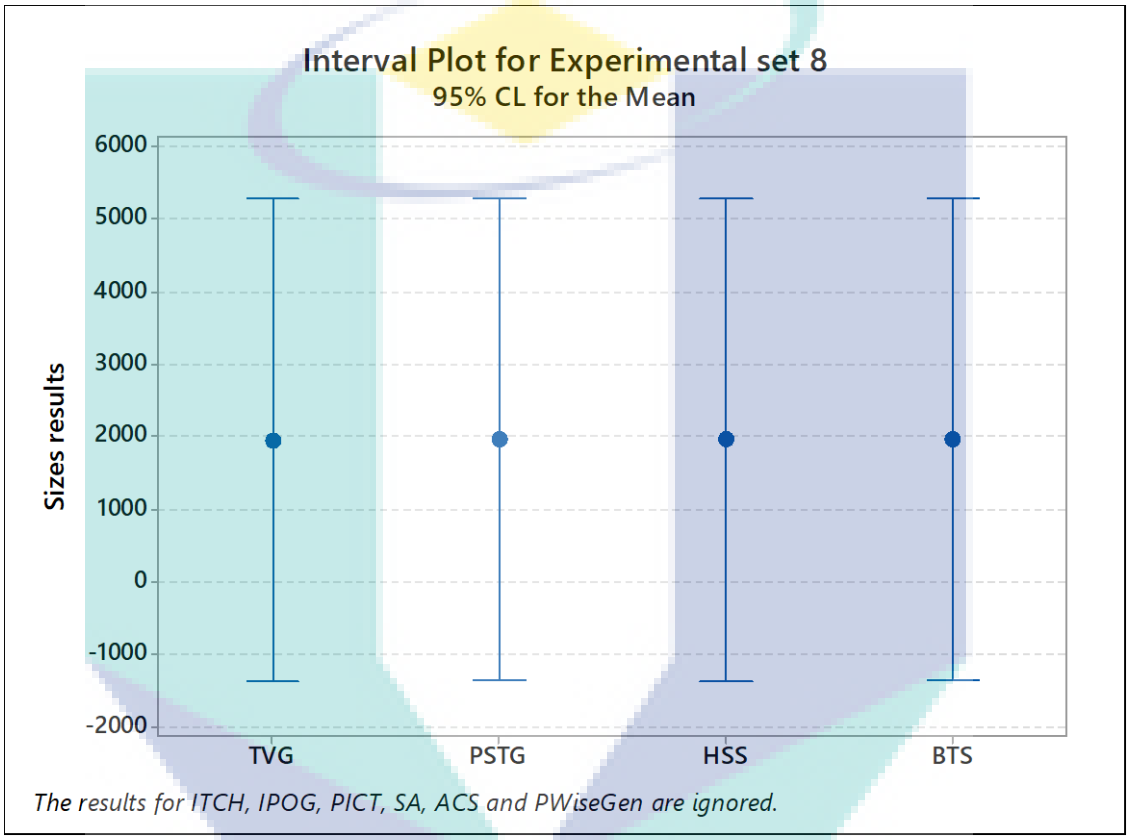


Figure 4.9 The illustration of Table 5.9 results' intervals with CL 95%.

4.4 Experimental Observation and Discussion

The statistical analysis of the mixed-strength benchmarking is reported in Tables 5.18 to 5.24 along with the four interval plots of the mean size distribution of each individual strategy results (test suite sizes). A general observation test (Friedman test) is reported first; then, in case of a statistical significant, a post-hoc test is performed. Figures 5.6 to 5.9 show each strategy's descriptive result distributions and the mean value.

4.4.1 Experimental Results and Statistical Analysis Observations

This section discusses the experimental results in details. Regarding the characterizing of BTS based on the comparison of the BTS variants (original BA and Hamming BA) reported in Table 4.1, the results of the benchmarking experiments revealed that the Hamming BA variant of BTS achieved the best sizes for the selected mixed covering arrays with variation of interaction strength from ($2 \leq t \leq 6$). The original BA variant of BTS matches the best result only in two entries (when $t = 3$ and $t = 5$). The same pattern can be seen as far as the average size is concerned. In terms of average time, the original BTS expectedly outperforms the Hamming based BTS. This performance is achieved owing to the overhead in implementing the Hamming distance classifier.

Concerning the convergence pattern based on the worst sizes in Figure 5.1, the BTS with Hamming BA shows higher converging rate (generate fewer test suite with fewer iterations) than the BTS with original BA in Experiment 1 and 5. For the other experiments, the BTS with Hamming BA is converging faster as well.

Regarding the second part of the benchmarking experiments in sub-section 4.2.2, the benchmarking results with other strategies (experimental sets 1 to 4 in Table 4.2 to Table 4.5) based on the Hamming BA variant of BTS show that the probabilistic strategies outperforms the deterministic strategies in general. From the statistical analysis of this sub-section, Friedman test gives a statistical significance for all the involved strategies in each experimental set (H_0 is rejected). Thus, a *post-hoc* test is performed to give a general observation for the median distribution of BTS against the other strategies (Table 4.2 to Table 4.5).

Based on the experimental sets 1, 2 and 3 The BTS is compared to TConfig, IPOG, ITCH, Jenny, PICT, TVG, PSTG, CS, and HSS strategies. For experimental set 1 in Table 4.2, the BTS excels in most of the test configurations with 77.5 % of the best (or most minimum) test suite sizes (31 out of 40 entries). Clearly, the BTS contributes to 25 out of the 31 entries with new minimal test suite sizes. In these test configurations, the BTS performed ahead of its counterparts especially when $t>3$. The ITCH and CS perform well in the low interaction strength ($t \leq 3$) with 10 and 8 entries (25 % and 20 of the most minimum test suite sizes, respectively). Here, the ITCH is not capable (or do not supported) of generating a test suite for high interactions ($t \geq 3$). Similar to ITCH and CS, the PSTG and HSS perform similarly with 4 entries each which is 10 % of the most minimum test suite sizes. Furthermore, Jenny and TVG only report the minimum test suite size of an entry. This is a 2.5 % of the minimal test suite sizes of the 40 entries. In these test configurations, IPOG, TConfig, and PICT have not reported significant or minimum test suite size for any entries.

From the statistical analysis shown in Table 4.2 (given in Table 4.10 and Table 4.11), Friedman test indicates that the null hypothesis is rejected at 95 % confidence level. Similarly, the post-hoc (Wilcoxon signed-rank) test in Table 4.11 shows that there is a significant difference. This is evidenced in the rejection of the null hypothesis for all pairs. All the BTS comparison with other strategies show differences as far as the size performance of BTS is concerned. Thus, the BTS is statistically better than other strategies based on the median distribution.

From the interval plots shown in Figure 4.2 for Table 4.2, it is noted that BTS manages to achieve the minimum overall average. The best performance is achieved based on the mean distribution at 95 % confidence levels. The HSS and PSTG show similar performance while CS is in the third rank. Jenny, TVG and PICT is ranked forth to sixth, respectively. Finally, IPOG shows the worst overall mean distribution, hence, having the poorest size performance. Here, TConfig and ITCH results are ignored owning to their result completeness (not complete sample).

According to the results of experimental set 2 in Table 4.3, the BTS manages to get the best results with a percentage entry of 60 % (12 out of 20 entries). It is interesting to note that 10 of the 12 best result entries are new minimal test suite sizes obtained by BTS. Similar to the observations in Table 4.2, the BTS excels in the high interaction

strength ($t > 3$). The other strategies shared the best results for 10 entries. Here, 2 of the 10 shared entries are with BTS. In these test configurations, HSS and CS perform well in 4 entries each while ITCH performed with 3 best sizes with low interaction strength ($t \leq 3$) followed by PSTG, IPOG and TConfig with 2 entries each. The rest of the strategies (Jenny, PICT and TVG) give similar results. However, no best sizes are obtained for Jenny, PICT and TVG.

Based on the statistical analysis shown in Table 4.3 (given in Table 4.12 and Table 4.13), the null hypothesis is rejected at 95 % confidence level by the Friedman test. As a result, there is a significant difference in terms of median distribution. Likewise, the *post-hoc* test indicates that the null hypothesis is rejected for all the pair compassion. The HSS is exempted here as far as size performance of BTS is concerned. For this reason, BTS is statistically better than other strategies in this experimental set with the exception of HSS.

The interval plot presented in Figure 4.3 for Table 4.3 shows that HSS gives the minimum mean. The BTS comes in second and the other strategies ranked in the following order; CS, PSTG, PICT, Jenny, TVG and IPOG (while ignoring the contributions of TConfig and ITCH).

Considering the results shown in Table 4.4, BTS, HSS and ITCH achieve the overall best test suite sizes. Clearly, the BTS achieves 50 % of the best sizes (3 out of 6 entries). Additionally, the obtained best sizes are the new best sizes generated by the BTS for the system configurations (P = 7, 8 and 9 with 1158, 1317 and 1508 test cases, respectively). The ITCH keeps the best sizes for 2 test configurations (P = 5 and 6, recording 625 test cases for each entry) with a percentage of 33.33 %. The HSS only gives the best result for one entry (P = 10 with 1624 test cases). For experimental set 3, only 3 strategies were observed to be able to achieve the best sizes. The other strategies produce acceptable results as compared to the best sizes.

In the statistical analysis presented in Table 4.4 (given in Table 4.14 and Table 4.15), Friedman test (Table 4.14) shows statistical significance at 95 % confidence level. Hence, the null hypothesis is rejected. As a result, a post-hoc test is considered. The post-hoc result shown Table 4.15 favors the alternate hypothesis in all the cases as far as median distribution is concerned. There is no statistical significant result based on the

pair comparisons). Thus, BTS is not significantly better than the other strategies in terms of median distribution of the sizes obtained.

From the mean distribution shown in the interval plot in Figure 4.4 for Table 4.4, the HSS is shown to give the minimum mean. The BTS follows in the second position. Overall, the performance of BTS is better than those of CS, PSTG, Jenny, PICT, ITCH, TVG, TConfig, and IPOG.

In experimental set 4, the BTS is executed against three available strategies (TConfig, PICT and TVG). Table 4.5 highlights the reported results of four real world open source software systems (Count, Nametbl, Flex, and Grap). The BTS manages to achieve a high size performance (best sizes) with a percentage of 88.89 % of the best test suite sizes (16 out of 18 entries). Additionally, the BTS also contributes to 13 minimal test suite sizes the other three entries shares with the other strategies (4-way Count and Flex, and 5-way Nametbl software). The PICT achieves 4 best test suite sizes with a percentage of 22.22 % (4 out of 18 system configurations for the 4-way and 5-way of Grap software). Here, the PICT achieves 2 minimal test suite sizes. The TVG achieves 2 best sizes (4-way Flex and 5-way Nametbl software). Finally, TConfig shares one best size with all of the strategies (5-way Nametbl software with a 450 test cases). Overall, the BTS dominates the best results in general. Most importantly, this new experimental set shows the size performance of BTS for real-world test suite generation.

From the statistical analysis shown in Table 4.5 (given in Table 4.16 and Table 4.17), Friedman test (Table 4.16) shows statistical significance at 95 % confidence level. Therefore, the null hypothesis is rejected. As a result, a *post-hoc* test is considered. The post-hoc analysis in Table 4.17 shows that the null hypothesis is rejected when the BTS pair is compared with TConfig and TVG. This indicates that BTS is significantly better than the TConfig and TVG. However, in the case of the BTS against PICT, the null hypothesis is accepted. Thus, there is no statistically significant result based on the comparison of BTS with PICT in terms of the median distribution of the obtained sizes.

From the illustration of the mean distribution shown in Figure 4.5 for Table 4.5, the BTS manages to be in the first rank with the minimum overall result average. Then, PICT, TConfig and TVG is ranked second, third and fourth, respectively.

Referring to the mixed-strength benchmarking experimental sets reported in sub-section 4.2.3 (experimental sets 5 to 8 in Table 4.6 to Table 4.9), It is generally observed that probabilistic based strategies outperforms the deterministic based strategies for both main-strength and sub-strength generations.

From the mixed-strength test configuration ($mCA\ (S;\ 2, 3^{15}, \{MC\}$), experimental set 5) shown in Table 4.6, the BTS performs well. Here, 50 % of the best sizes are obtained (9 out of 18 entries). The BTS manages to obtain three new minimal test suite sizes (in the following sub-configurations; $CA\ (S, 4, 3^5)$, $CA\ (S, 4, 3^7)$, and $CA\ (S, 5, 3^7)$). Thus, an improvement of 16.66 % (3 out of 18 new minimum test suite) of the total entries is achieved. For the other best sizes, the BTS shares 33.33 % with other strategies. In these test configurations, the SA-Mayer generates the best test suite in general with a percentage of 61.11% (11 out of 18 entries). Specifically, the SA-Mayer in the low interaction strength achieved the best results as it only supports *t* up to 3 ($t \leq 3$). Putting the SA-Mayer aside, the PWiseGen-VSCA, HSS, and VS-PSTG produce competitive results as well with percentages of 50 %, 44.45 % and 33.33 % best sizes. This corresponds to 9, 8, and 6 out of 18 entries, respectively. For the ACS-VSITs, TVG and IPOG performs similarly with 22.22 % of the best sizes obtained (4 out of 18 entries each). The PICT and ITCH generates the worst results for the mixed-strength test configuration.

From the statistical analysis shown in Table 4.6 (given in Table 4.18 and Table 4.19), the null hypothesis is rejected at 95 % confidence level (Friedman test in Table 4.18). Then, the post-hoc test in Table 4.19 indicates that the BTS is statistically better than other strategies. This is based on the median distribution with the exception of HSS. Unlike the BTS against HSS (null hypothesis accepted), the null hypothesis is rejected for BTS against ITCH, PICT, TVG, PSTG. The contributions of IPOG, SA-Mayer, ACS-VSITs, and PWiseGen-VSCA are ignored. Thus, the BTS is statistically better than ITCH, PICT, TVG and PSTG based on the median distribution.

The interval plot presented in Figure 4.6 for Table 4.6 shows that the HSS gave the minimum mean. The BTS comes in the second rank followed by the other strategies ranked in in the following order; PSTG, TVG, ITCH and PICT. PICT performs the worst in terms of the overall mean distribution of results.

From experimental set 6, the mixed-strength test configuration $mMCA\ (S;\ 2, 4^3 5^3 6^2, \{MC\})$ in Table 4.7, SA-Mayer, and HSS perform best with a percentage of 61.54 %. This is the best result accounting for 8 out of 13 entries each). The BTS, VS-PSTG, and PWiseGen-VSCA perform well with a percentage of 53.85 % of the best test suite sizes. This corresponds to 7 out of 13 entries each. The BTS manages to obtain a new minimal test suite size in the case of $MCA(S, 4, 4^3 5^2)$ sub-strength configuration with 2,380 test cases. Here, the BTS, VS-PSTG, and PWiseGen-VSCA generate the same results for the low interaction strength ($t \leq 3$). For the interaction strength values ($t > 3$), the HSS and BTS outperformed all the other strategies. The TVG, ACS-VSITs, and IPOG also generated competitive results with a percentage of 46.15 %, 30.77 % and 15.38 % of the best results, respectively. The ITCH and PICT consistently produce the worst results for the mixed-strength test configuration.

From the statistical analysis in Table 4.7 (given in Table 4.20 and Table 4.21), the Friedman test in Table 4.20 favoured the null hypothesis at 95 % confidence level. However, the post-hoc test shown in Table 4.21 indicates that BTS is not statistically better than the other strategies based on the median distribution. As a result, the null hypothesis is retained for all the pair comparisons). The contribution of IPOG, PICT, SA-Mayer and ACS-VSITs is ignored due to incomplete samples.

The interval plot presented in Figure 4.7 for Table 4.7 shows that HSS gives the minimum mean. The BTS comes in the second rank and the other strategies ranked in the following order; PSTG, PWiseGen-VSCA, TVG and ITCH.

In experimental set 7, the mixed-strength test configuration given in Table 4.8 demonstrates acceptable performance of several strategies (BTS, HSS, VS-PSTG, PWiseGen-VSCA, TVG and IPOG) for this test generation. It is observed that these strategies perform well with the increased number of parameter values as in the test configuration. In the case of the mixed-strength test configuration ($mMCA\ (S;\ 2, 10^1\ 9^1\ 8^1\ 7^1\ 6^1\ 5^1\ 4^1\ 3^1\ 2^1,\ \{\text{MC}\})$), the BTS excels in most cases with a percentage of 81.82 % of the best results (9 out of 11 entries). Furthermore, the BTS manages to get a new minimal test suite size for one sub-strength ($MCA\ (S, 3, 10^1 9^1 8^1 7^1)$). Regarding HSS and VS-PSTG, these strategies perform equally with a percentage of 72.72 % of the best test suite sizes (8 out of 11 entries). Similarly, PWiseGen-VSCA and TVG obtain a percentage of 63.63 % of the best sizes

(7 out of 11 entries). In the same manner, the IPOG generates competitive test suite size in many sub-strength configurations with a percentage of 54.55 % of the best sizes. The ITCH and PICT generate the poorest results in most cases with no best size obtained among all the test configurations (with some missing results). As for SA-Mayer and ACS-VSITs, no published results are available.

From the statistical analysis in Table 4.8 (given in Table 4.22 and Table 4.23), Friedman test indicates that there is a significant difference at 95 % confidence level. The null hypothesis is thereby, rejected. The *post-hoc* test in Table 4.23 shows that there is only statistically significant difference in the case of BTS against ITCH. The null hypothesis is also rejected. Nevertheless, the other pair comparisons in the *post-hoc* test retain the null hypothesis. Thus, the performance of BTS is only statistically better than that of ITCH. Here, the results of PICT, SA-Mayer, ACS-VSITs and PWiseGen-VSCA are ignored.

The interval plots presented in Figure 4.8 for Table 4.8 show that HSS, PSTG, TVG, IPOG and BTS is ranked first while ITCH ranks last. Here, the BTS performs in a similar way to the other strategies in terms of mean distribution.

In Table 4.9, the results of the experimental set 8 are reported. Here, the mixed-strength test configuration ($mMCA\ (S;\ 2, 3^{20}10^{2}, \{MC\})$ is benchmarked with a sub-strength up to t = 6 (high sub-strength interaction). The BTS, HSS, VS-PSTG, SA-Mayer, IPOG and TVG are able to obtain a percentage of 50 % of the best sizes each (3 out of 6 entries each). In fact, the BTS, HSS and VS-PSTG generate the optimal test suite size for high interaction strength ($t > 3$). However, the SA-Mayer dominates the low interaction strength ($t \leq 3$). The ACS-VSITs strategy obtains with a 33.33 % of the best sizes (2 out of 6 entries). The PICT achieves16.66 % of the best sizes (1 out of 6 entries). Regarding PWiseGen-VSCA and ITCH, no published results are available.

From the statistical analysis shown in Table 4.9 (given in Table 4.24), the general observation test (Friedman test) favours the alternative hypothesis at 95 % confidence level. The null hypothesis is retained and as a result, a *post-hoc* test is not considered. This is because there is no statistically significant difference between the BTS against HSS, PSTG and TVG. The contribution of TCH, IPOG, PICT, SA-Mayer, ACS-VSITs and PWiseGen-VSCA are ignored.

From the interval plots shown in Figure 4.9 for Table 4.9, the BTS, HSS, PSTG and TVG ranks in similar manner as far as mean distribution of their overall results is concerned.

4.4.2 Discussion

Overall, the BTS gives competitive test suite sizes in most of the test configurations considered. The BTS manages to achieve the optimal test suite (new minimal test suite size) in a number of test configurations as detailed in the previous sub-section.

From the first part of the comparative benchmarking in section 4.2.1, the Hamming BA variant achieves better results than the original BA (see Table 4.1). However, the original BA has a lower execution time as compared to the Hamming BA for all the specified configurations.

At a glance, the convergence pattern of experiment 2 to 4 looks similar. However, a closer look revealed otherwise. Unlike experiment 1 and 5, the Hamming BA converges faster (in the smallest test suite the worst-case scenario) at less iteration (see Figure 5.1). Here, it is observed that the Hamming BA outperforms the original BA in terms of size performance. Here, this convergence pattern works well owing to the adoption of Hamming distance classifier. Specifically, the Hamming distance classifier improves the exploration of Hamming BTS. The exploration roams the random search space on a global scale (BA global search) by selecting the highest distance test case from the pool of generated candidates when there is a tie between 2 or more test candidates. In effect, the Hamming BTS ensures wider coverage of test cases with more diversity. On the negative note, the exploration consumed more time and computational resources owing to the need to compute and evaluate the Hamming distance.

Contrarily, the exploitation in both the original BA and Hamming BA focuses on searching in a local region via exploiting the current suitable solution (through BA random walks). The extreme exploitation tends to reject the diversified solutions and led to local optima. It is the loudness and emission of pulse rates that controls the BA exploitation using random walks. As no change is made in our Hamming BA as far as

loudness and emission of pulse rate is concerned, the implementation of the Hamming distance classifier did not affect the exploitation.

Based on the eight experimental benchmarked experiments in Section 4.2.2 and 4.2.3, the BTS obtains competitive results in all the cases. Table 4.25 reports the percentage of the number of best sizes (the minimal sizes) obtained by the BTS. The number is out of the total number of benchmarked system configurations for each of the conducted experimental sets. To be specific, the BTS achieves 68.181% of the best sizes of published results (90 out of 132 entries). The BTS contributes 32.575 % corresponding to 43 out of 90 best sizes (new obtained best sizes).

Table 4.25 The experimental sets observation.

Experimental sets	**Number of Improved best sizes by BTS**	**Number of best sizes obtained**	**Total system configurations benchmarked**	**The percentage of the Improved best sizes**	**The percentage of the obtained best sizes**
1	25	31	40	62.500%	77.500%
2	10	12	20	50.000%	60.000%
3	3	3	6	50.000%	50.000%
4	*	16	18	*	88.888%
5	3	9	18	16.666%	50.000%
6	1	7	13	7.6923%	53.846%
7	1	9	11	9.0909%	81.818%
8	0	3	6	0.0000%	50.000%
Total	**43**	**90**	**132**		

*Note: Experimental set 4 is a new set of experiments. Thus, no new best is considered (i.e. no improved best sizes as its for this study).

From the statistical analysis reported in Table 4.26, the BTS offers statistical significance in 20 out of 41 cases considered (48.78 %) at 95 % confidence level.

Table 4.26 The statistical significant achieved for each experimental sets.

Experimental Sets	**Comparison to the all Strategies**	**Statistical Significant Comparison**	**Number of Statistical Significant Cases**
1	All strategies	BTS vs IPOG	7 out of 7
		BTS vs PICT	
		BTS vs Jenny	
		BTS vs TVG	
		BTS vs HSS	
		BTS vs PSTG	
		BTS vs CS	
2	Except HSS	BTS vs PICT	6 out of 7
		BTS vs Jenny	
		BTS vs TVG	
		BTS vs IPOG	
		BTS vs PSTG	
		BTS vs CS	
3	None (no statistical significant)		0 out of 9
4	Except PICT	BTS vs TConfig	2 out of 3
		BTS vs TVG	
5	Except HSS	BTS vs ITCH	4 out of 5
		BTS vs PICT	
		BTS vs TVG	
		BTS vs PSTG	
6	None		0 out of 5
7	Except TVG, PSTG, HSS and IPOG	BTS vs ITCH	1 out of 5
8	None, *Post-hoc* test has not performed		-
Total			20 out of 41

Referring to Table 4.26, the null hypotheses is rejected for 20 pair comparisons. In the case of experimental set 1, all the comparisons are in favour of BTS (7 out of 7 entries). In experimental sets 2, 4 and 5, the null hypothesis is rejected except in the case of one strategy (HSS for experimental sets 2 and 5, PICT for experimental sets 4). In experimental set 7, only the case with ITCH showed statistical significance. In experimental sets 3 and 6, no statistical significance is achieved (the null hypothesis is retained in all the pair comparisons). Similarly, there is no statistical significance is achieved in experimental set 8.

4.5 Summary

This chapter has presented the performance of BTS. The benchmarking of BTS includes comparative evaluation against existing strategies as well as the corresponding statistical analysis. The size performance of BTS achieved statistically significant results.

Building on the current content in this chapter, the next chapter will summarize all the findings, make conclusions and remark on contributions, as well as provide a roadmap for possible future research in this direction.

CHAPTER 5

CONCLUSION AND FUTURE WORK

The previous chapter has subjected BTS with a number of experiments in order to establish its true performance in terms of generated test suite size. Building from all the materials presented in the previous chapters, this chapter highlights the impact of the results obtained and implication for future work.

5.1 Objectives Revisited

This research effort was aimed to design, implement and evaluate a *t*-way test generation strategy that supports mixed-strength interaction, called Bat-inspired *t*-way Strategy (BTS). The objectives of this research study were as follows:

- To design BTS strategy for constructing a mixed strength *t*-way test suite.
- To implement BTS as a research prototype using BA as the backbone search engine and introduces Hamming distance classifier in order to enhance the exploration of BA.
- To evaluate the test suite size performance of BTS against existing strategies using well-known benchmarking case studies.

Addressing the first objective, a new t-way test suite generation strategy, called BTS, is developed. The proposed strategy is designed to generate a minimized t-way and mixed-strength test suite taking the BA algorithm as the basis of the study. The main key aspect of this objective is satisfied owing to the successful implementation of BTS.

The BTS strategy generates a mixed *t*-way test suite, within the objective of generating the minimal test suite that is valid and covers all the possible test configurations up to $t = 6$. The BTS strategy takes on the test configurations (expressed

as covering array notation) for software under test. The strategy processes the test specification requirement as notation to minimize the test suite automatically.

Concerning the second objective, BTS employs the BA as a search engine to find the maximum covered interaction by each generated test case. BA is implemented in BTS to support the optimal finding for the combinatorial interaction test generation. BA is also modified to fit our implementation. Additionally, BA is enhanced using a best selection technique through the Hamming distance classifier. Specifically, the exploration process of BA global search is improved.

BTS provides execution scalability for test suite execution approaches. As the data size increases rapidly based on increasing test parameters and their dependencies, testing all the data becomes difficult and resource consuming. Addressing this limitation, BTS provides the ability to store an optimal test suite into files for the execution process. In this manner, test engineers can execute the testing activity without having to think about the correctness of their test cases as BTS generated test data covers all test data, hence, potentially saving resources and time. Furthermore, BTS generated test suite can be easily integrated with automated test execution approaches (i.e. the output is a simple file that lists the entire generated test cases).

As for the final objective, BTS has been successfully employed to undertake all the experimentations given, hence, highlighting its size performance for test suite generation. Experimentation against several well-known strategies have helped to reveal the performance of BTS in a seamless manner. In the conducted evaluation, BTS results are successfully compared against the available *t*-way test suite generation strategies. BTS experimental results have been encouraging as many newly introduced *t*-way results.

Considering the mixed-strength test configurations (i.e. system with multiple numbers of parameter and values) and highly interacting test configurations (i.e. for $t > 3$), BTS often produces the optimal test suite. As for mixed-strength test suite, BTS consistently generates the best test suite size. Overall, BTS obtained 68.181% of the best sizes of all the benchmarked test configurations (i.e. 90 out of 132 entries) as reported in Table 4.25. Additionally, BTS manages to improve the best sizes published in the literature (Ahmed et al., 2015; Ahmed et al., 2014; Ahmed & Zamli, 2011b; Bansal et al.,

2015; Cohen, 2004; Cohen et al., 2003c; Lei et al., 2007; Shiba et al., 2004; Wang & He, 2013; Xiang et al., 2009) with a 32.575% (i.e. 43 out of 132 entries (see Table 4.2 to Table 4.4 and Table 4.6 to Table 4.9)). Furthermore, the statistical analysis shows 48.78% statistical significance based on the pier compression of Wilcoxon signed-rank (see Table 4.26). Therefore, this study concludes that that BTS is a useful strategy for generating *t*-way and mixed-strength test suites.

In this thesis, BTS strategy is designed for test suite generation of high inputs (highly configurable) software system. The generation method support generating high scale input (i.e. 2^{100} and more) as the BTS uses the advantage of Java for memory management to efficiently utilize the available memory for the generated search space. The use of string interaction elements and test candidates reduce the memory needed for high configuration system as string variable can store a lot of information that can be parsed in the test generation process. This allows BTS to generate a test site for highly configurable software systems.

5.2 Contribution

Summing up, based on earlier discussion, the main contribution of BTS relates to its *t*-way test suite generation support. The research contribution undertaken in this research work can be stated from different perspectives as follows:

- BTS is the first strategy that applies BA as a backbone engine to its test suite reduction mechanism.
- BTS modified the BA algorithm by employing a selection (or best finding) technique that improves the exploration of the BA (i.e. improving the global random search) via the Hamming distance classifier.
- BTS contributes to a number of well-known benchmarking test configurations published literatures with 43 new test suite sizes (see Table 4.2 to Table 4.9).

5.3 Future work

Given that the application of BTS presented in this study is still a prototype, an obvious starting point for future work will be to complete the implementation to support automated test execution and other *t*-way test generation types. In particular, several *t*-way features needed to be included (i.e. input-output relations *t*-way, sequencing *t*-way and constraints *t*-way).

Currently, BTS only addresses the automated test suite generation. Therefore, in order to improve its applicability, there is also a need to automate as much as possible the execution of test cases generated by BTS whenever possible. This automation could be in the form of automatic translation of the test cases generated by BTS into actual executable form through some forms of scripting language. Such endeavour will help alleviate the burden of test engineers from cumbersome and manual test execution.

Providing constraints support for BTS including Software Product Line (SPL) is one area for exploration. BTS strategy can be modified to add constraints interaction support. The constraints interaction can be established by removing the constraints configurations (or invalid configurations) from the IET and re-generate the test cases that involved constraints. Expectedly, a similar reduction of test cases can be achieved.

Finally, sequence support to BTS strategy could also be explored. In some domain implementation, sequences of input parameters do matters. Thus, it is desirable for BTS to be able to provide sequence-based *t*-way test suite generation.

REFERENCES

Afzal, W., Torkar, R., and 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., Abdulsamad, T. S., and Potrus, M. Y. 2015. Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the cuckoo search algorithm. *Information and Software Technology*. **66**(0): 13-29.

Ahmed, B. S., Sahib, M. A., and Potrus, M. Y. 2014. Generating combinatorial test cases using simplified swarm optimization (SSO) algorithm for automated GUI functional testing. *International Journal Engineering Science and Technology*. **17**(4): 218-226.

Ahmed, B. S., and Zamli, K. Z. 2010a. PSTG: A t-way strategy adopting particle swarm optimization. *In the 4th Asia International Conference on Mathematical /Analytical Modelling and Computer Simulation*, 1-5.

Ahmed, B. S., and Zamli, K. Z. 2010b. t-way test data generation strategy based on particle swarm optimization. *In the 2nd International Conference on Computer Research and Development*, 93-97.

Ahmed, B. S., and Zamli, K. Z. 2011a. The development of a particle swarm based optimization strategy for pairwise testing. *Journal of Artificial Intelligence*. **4**(2): 156-165.

Ahmed, B. S., and Zamli, K. Z. 2011b. A variable-strength interaction test suites generation strategy using particle swarm optimization. *Journal of Systems and Software*. **84**(12): 2171-2185.

Ahmed, B. S., Zamli, K. Z., and Lim, C. P. 2012a. Application of particle swarm optimization to uniform and variable strength covering array construction. *Applied Soft Computing*. **12**(4): 1330-1347.

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*. **8**(1): 431-452.

Ali, E. S. 2014. Optimization of power system stabilizers using BAT search algorithm. *International Journal of Electrical Power and Energy Systems*. **61**(1): 683-690.

Ali, S., Briand, L. C., Hemmati, H., and Panesar-Walawege, R. K. 2010. A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Transactions on Software Engineering*. **36**(6): 742-762.

Alsewari, A. A., and Zamli, K. Z. 2012a. 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.

Alsewari, A. A., and Zamli, K. Z. 2012b. A harmony search based pairwise sampling strategy for combinatorial testing. *International Journal of The Physical Sciences*. **7**(7): 1062-1072.

Alsewari, A. A., and Zamli, K. Z. 2014. An orchestrated survey on t-way test case generation strategies based on optimization algorithms. *In the 8th International Conference on Robotic, Vision, Signal Processing and Power Applications*, **291**, 255-263.

Alsewari, A. A., Zamli, K. Z., and Al-Kazemi, B. 2014. Generating t-way test suite in the presence of constraints. *In the 8th Malaysia University Conference Engineering Technology*.

Alsewari, A. A., Zamli, K. Z., and Al-Kazemi, B. 2015. Generating t-way test suite in the presence of constraints. *Journal of Engineering and Technology*. **6**(2): 52-66.

Ammann, P., and Offutt, J. 1994. Using formal methods to derive test frames in category-partition testing. *In the 9th Annual Conference on Computer Assurance*, 69-80.

Antony, J. 2003. *Design of experiments for engineers and scientists*. 1st ed. India: Elsevier Science and Technology Books.

Arshem, J. 2003. Test vector generator (TVG), Available from: http://sourceforge.net/ projects/tvg, last accessed on (November, 2016).

Bach, J. 2002. Allpairs test case generation tool, version 1.2.1, Available from: http://www.satisfice.com/tools.shtml, last accessed on (November, 2016).

Bansal, P., Sabharwal, S., Mittal, N., and Arora, S. 2015. Construction of variable strength covering array for combinatorial testing using a greedy approach to genetic algorithm. *E-Informatica Software Engineering Journal*. **9**(1): 87-105.

Bansal, P., Sabharwal, S., Mittal, N., and Arora, S. 2016. ABC-CAG: Covering array generator for pair-wise testing using artificial bee colony algorithm. *E-Informatica Software Engineering Journal*. **1**(10): 9-29.

Bao, X., Liu, S., Zhang, N., and Dong, M. 2015. Combinatorial test generation using improved harmony search algorithm. *International Journal of Hybrid Information Technology*. **8**(9): 121-130.

Baresi, L., and Pezzè, M. 2006. An introduction to software testing. *Electronic Notes in Theoretical Computer Science*. **148**(1): 89-111.

Bertolino, A. 2007. Software testing research: achievements, challenges, dreams. *In the Future of Software Engineering*, 85-103.

Bryce, R. C., and Colbourn, C. J. 2007. One-test-at-a-time heuristic search for interaction test suites. *In the 9th Annual Conference on Genetic and Evolutionary Computation*, 1082-1089.

Bryce, R. C., Colbourn, C. J., and Cohen, M. B. 2005. A framework of greedy methods for constructing interaction test suites. *In the 27th International Conference on Software Engineering*, 146-155.

Burnstein, I. 2006. *Practical software testing*. 1st ed. Chicago: Springer Science and Business Media Inc.

Burr, K., and Young, W. 1998. Combinatorial test techniques: table-based automation, test generation and code coverage. *In the International Conference on Software Testing Analysis and Review*, 1-12.

Burroughs, K., Jain, A., and Erickson, R. L. 1994. Improved quality of protocol testing through techniques of experimental design. *In the International Conference on Communications, Serving Humanity Through Communications*, **2**, 745-752.

Bush, K. A. 1952. Orthogonal arrays of index unity. *The Annals of Mathematical Statistics*. **23**(3): 426-434.

Carroll, C. T. 2003. The cost of poor testing: a us government study (part 1). *EDPACS: The EDP Audit, Control, and Security Newsletter*. **31**(2): 1-17.

Chaudhuri, D. K. R., and Zhu, T. 1992. A recursive method for construction of designs. *Discrete Mathematics*. **106**(107): 399-406

Chen, S.-M., and Chien, C.-Y. 2011. Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. *Expert Systems with Applications*. **38**(12): 14439-14450.

Chen, W.-N., and Zhang, J. 2009. An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements. *IEEE Transactions on Systems, Man and Cybernetics*. **39**(1): 29-43.

Chen, X., Gu, Q., Qi, J., and Chen, D. 2010. Applying particle swarm optimization to pairwise testing. *In the 34th Annual Computer Software and Applications Conference*, 107-116.

Chen, X., Gu, Q., Zhang, X., and Chen, D. 2009. Building prioritized pairwise interaction test suites with ant colony optimization. *In the 9th International Conference on Quality Software*, 347-352.

Cheng, C.-S. 1980. Orthogonal arrays with variable numbers of symbols. *The Annals of Statistics*. **8**(2): 447-453.

Cohen, D. M. 2011. AETG Web, Available from: http://aetgweb.argreenhouse.com/pricing.shtml, last accessed on (November, 2016).

Cohen, D. M., Dalal, S. R., Fredman, M. L., and Patton, G. C. 1997. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*. **23**(7): 437-444.

Cohen, D. M., Dalal, S. R., Kajla, A., and Patton, G. C. 1994. The automatic efficient test generator (AETG) system. *In the 5th International Symposium on Software Reliability Engineering*, 303-309.

Cohen, D. M., Dalal, S. R., Parelius, J., and Patton, G. C. 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*. Ph.D. Thesis. University of Auckland, New Zealand.

Cohen, M. B., Colbourn, C. J., Gibbons, P. B., and Mugridge, W. B. 2003a. Constructing test suites for interaction testing. *In the 25th IEEE International Conference on Software Engineering*, 38-48.

Cohen, M. B., Colbourn, C. J., and Ling, A. C. 2008a. Constructing strength three covering arrays with augmented annealing. *Discrete Mathematics.* **308**(13): 2709-2722.

Cohen, M. B., Colbourn, C. J., and Ling, A. C. H. 2003b. Augmenting simulated annealing to build interaction test suites. *In the 14th International Symposium on Software Reliability Engineering*, 394-405.

Cohen, M. B., Dwyer, M. B., and Jiangfan, S. 2007a. Exploiting constraint solving history to construct interaction test suites. *In the Testing: Academic and Industrial Conference Practice and Research Techniques*, 121-132.

Cohen, M. B., Dwyer, M. B., and Shi, J. 2007b. Interaction testing of highly-configurable systems in the presence of constraints. *In the International Symposium on Software Testing and Analysis*, 129-139.

Cohen, M. B., Dwyer, M. B., and Shi, J. 2008b. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Transactions on Software Engineering.* **34**(5): 633-650.

Cohen, M. B., Gibbons, P. B., Mugridge, W. B., Colbourn, C. J., and Collofello, J. S. 2003c. Variable strength interaction testing of components. *In the 27th Annual International Computer Software and Applications Conference*, 413-418.

Colbourn, C. J. 2009. CA tables, Available from: http://www.public.asu.edu/~ccolbou/src/tabby/catable.html, last accessed on (November, 2016).

Colbourn, C. J. 2011. *Covering arrays and hash families*. IOS Press.

Colbourn, C. J., and Dinitz, J. H. 2006. *Handbook of combinatorial designs (Discrete Mathematics and Its Applications)*. 2nd ed.: Chapman and Hall CRC press.

Copeland, L. 2004. *A practitioner's guide to software test design*. 1st ed. Boston: Artech House Inc.

Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., and Wąsowski, A. 2012. Cool features and tough decisions: a comparison of variability modeling approaches. *In the 6th International Workshop on Variability Modeling of Software-Intensive Systems*, 173-182.

Czerwonka, J. 2006. Pairwise testing in the real world: practical extensions to test-case scenarios. *In the 24th Pacific Northwest Software Quality Conference*, **82**, 419-430.

Daich, G. T. 2003. Testing combinations of parameters made easy. *In the IEEE Systems Readiness Technology Conference*, 379-384.

Dalal, S. R., Jain, A., Karunanithi, N., Leaton, J. M., Lott, C. M., Patton, G. C., and Horowitz, B. M. 1999. Model based testing in practice. *In the International Conference on Software Engineering*, 285–294.

Dalal, S. R., Karunanithi, A. J., N. , Leaton, J. M., and Lott, C. M. 1998. Model-based testing of a highly programmable system. *In the 9th International Symposium on Software Reliability Engineering*, 174–178.

Daniel, W. W. 1990. *Friedman two-way analysis of variance by ranks*. 2nd ed. Boston: PWS-Kent.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation.* **6**(2): 182-197.

Dias Neto, A. C., Subramanyan, R., Vieira, M., and Travassos, G. H. 2007. A survey on model-based testing approaches: a systematic review. *In the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages*, 31-36.

Dorigo, M., Birattari, M., and Stutzle, T. 2006. Ant colony optimization. *IEEE Computational Intelligence Magazine.* **1**(4): 28-39.

Dorigo, M., Maniezzo, V., and Alberto, C. 1989. The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics.* **26**(1): 1-13.

Drigo, M., Maniezzo, V., and Colorni, A. 1996. Ant system: optimization by a colony of cooperation agents. *IEEE Transactions on Systems, Man and Cybernetics.* **26**(1): 29-41.

Duran, J. W., and Ntafos, S. C. 1984. An evaluation of random testing. *IEEE Transactions on Software Engineering.* **10**(4): 438-444.

Ellims, M., Ince, D., and Petre, M. 2008. AETG vs. Man: an assessment of the effectiveness of combinatorial test data generation UK:

Flores, P., and Cheon, Y. 2011. Pwisegen: Generating test cases for pairwise testing using genetic algorithms. *In the International Conference on Computer Science and Automation Engineering*, **2**, 747-752.

Forbes, M., Lawrence, J., Lei, Y., Kacker, R. N., and Kuhn, D. R. 2008. Refining the in-parameter-order strategy for constructing covering arrays. *Journal of Research of the National Institute of Standards and Technology.* **113**(5): 287-297.

Gandomi, A. H., Yang, X.-S., Alavi, A. H., and Talatahari, S. 2013. Bat algorithm for constrained optimization tasks. *Neural Computing and Applications.* **22**(6): 1239-1255.

Garvin, B. J., Cohen, M. B., and Dwyer, M. B. 2009. An improved meta-heuristic search for constrained interaction testing. *In the 1st International Symposium on Search Based Software Engineering*, 13-22.

Garvin, B. J., Cohen, M. B., and Dwyer, M. B. 2011. Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering.* **16**(1): 61-102.

George, H. A. 2012. *Constructing covering arrays using parallel computing and grid computing.* Ph.D. Thesis. Universitat Politecnica de Valencia, Spain.

Gherbi, Y. A., Bouzeboudja, H., and Lakdja, F. 2014. Economic dispatch problem using bat algorithm. *Leonardo Journal of Sciences.* **13**(24): 75-84.

Gonzalez-Hernandez, L. 2015. New bounds for mixed covering arrays in t-way testing with uniform strength. *Information and Software Technology.* **59**(0): 17-32.

Gonzalez-Hernandez, L., Rangel-Valdez, N., and Torres-Jimenez, J. 2010. Construction of mixed covering arrays of variable strength using a tabu search approach. *In the International Conference on Combinatorial Optimization and Applications*, 51-64.

Grindal, M., Offutt, J., and Andler, S. F. 2005. Combination testing strategies: a survey. *Software Testing, Verification and Reliability.* **15**(3): 167-199.

Hartman, A., Klinger, T., and Raskin, L. 2005. WHITCH: IBM intelligent test configuration handler IBM Haifa and Watson Research Laboratories: September.

Hartman, A., and Raskin, L. 2004a. Combinatorial test services, Available from: https://www.research.ibm.com/haifa/projects/verification/mdt/tools.html, last accessed on (August, 2016).

Hartman, A., and Raskin, L. 2004b. Problems and algorithms for covering arrays. *Discrete Mathematics.* **284**(1): 149-156.

Haslinger, E. N., Lopez-Herrejon, R. E., and Egyed, A. 2013. Improving casa runtime performance by exploiting basic feature model analysis. *Arxiv Preprint Arxiv:1311.7313.*

Hass, A. M. 2008. *Guide to advanced software testing* 2nd ed. Norwood: Artech House Inc.

Hegazy, O., Soliman, O. S., and Salam, M. A. 2015. Comparative study between FPA, BA, MCS, ABC, and PSO algorithms in training and optimizing of LS-SVM for stock market prediction. *International Journal of Advanced Computer Research.* **5**(18): 35.

Huang, S., Cohen, M. B., and Memon, A. M. 2010. Repairing GUI test suites using a genetic algorithm. *In the 3rd International Conference on Software Testing, Verification and Validation*, 245-254.

Inc., S. T. 2014. SmartTest - pairwise testing tool (Smartware Technologies Inc.), Available from: http://www.smartwaretechnologies.com/smarttestprod.htm, last accessed on (July, 2016).

Jia-Ze, S., and Shu-Yan, W. 2012. *Generation of pairwise test sets using a novel DPSO algorithm*. Springer.

Kaner, C., Falk, J., and Nguyen, H. Q. 1999. *Testing computer software*. 2nd ed. New York: Dreamtech Press.

Katherine, A. V., and Alagarsamy, D. K. 2012. Conventional software testing vs. Cloud testing. *International Journal of Scientific and Engineering Research.* **3**(9): 1-5.

Khalsa, S. K., and Labiche, Y. 2014. An orchestrated survey of available algorithms and tools for combinatorial testing. *In the 25th International Symposium on Software Reliability Engineering*, 323-334.

Khan, K., and Sahai, A. 2012. A comparison of BA, GA, PSO, BP and LM for training feed forward neural networks in e-learning context. *International Journal of Intelligent Systems and Applications.* **4**(7): 23.

Khatun, S., Rabbi, K. F., Yaakub, C. Y., Klaib, M. F., and Ahmed, M. M. 2011. PS2Way: An efficient pairwise search approach for test data generation. *In the International Conference on Software Engineering and Computer Systems*, 99-108.

Klaib, M. F. 2009. *Development of an automated test data generation and execution strategy using combinatorial approach.* Ph.D. Thesis. Universiti Sains Malaysia, USM.

Klaib, M. F., Zamli, K. Z., Isa, N. A. M., Younis, M. I., and Abdullah, R. 2008. G2Way – a backtracking strategy for pairwise test data generation. *In the 15th Asia-Pacific Software Engineering Conference*, 463-470.

Klaib, M. F. J., Al-batah, M. S., and Rasras, R. J. 2015. 3-way interaction testing using the tree strategy. *Procedia Computer Science.* **65**(1): 845-852.

Krishnan, R., Krishna, S. M., and Nandhan, P. S. 2007. Combinatorial testing: Learnings from our experience. *ACM Software Engineering Notes.* **32**(3): 1-8.

Kuhn, D. R., Dolores, R. W., and Gallo, A. M. 2004. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering.* **30**(6): 418-421.

Kuhn, D. R., Kacker, R. N., and Lei, Y. 2010. Practical combinatorial testing. *NIST Special Publication.* 1-75.

Kuhn, D. R., Lei, Y., and Kacker, R. 2008. Practical combinatorial testing: beyond pairwise. *IEEE IT Professionals.* **10**(3): 19-23.

Kuhn, D. R., NIST, R. N. K., and NIST, Y. L. 2015. Combinatorial coverage as an aspect of test quality. *Crosstalk.* **28**(2): 19-23.

Kuliamin, V. V., and Petukhov, A. 2011. A survey of methods for constructing covering arrays. *Programming and Computer Software.* **37**(3): 121-146.

Laerd Statistics, W. 2017a. Friedman test in SPSS statistics, Available from: https://statistics.laerd.com/spss-tutorials/friedman-test-using-spss-statistics.php, last accessed on (January, 2017).

Laerd Statistics, W. 2017b. Wilcoxon signed-rank test using SPSS statistics, Available from: https://statistics.laerd.com/spss-tutorials/wilcoxon-signed-rank-test-using-spss-statistics.php, last accessed on (January, 2017).

Lehmann, E., and Wegener, J. 2000. Test case design by means of the CTE XL. *In the 8th European International Conference on Software Testing, Analysis and Review*, 1-10.

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

Lei, Y., and Tai, K. C. 1998. In-Parameter-Order: a test generation strategy for pairwise testing. *In the 3rd IEEE International Symposium on High-Assurance Systems Engineering* 254–261.

LI, J., XING, D., and ZHAO, Y. 2013. Combinatorial test suite generation of variable strength based on harmony search. *Journal of Network and Information Security*. **4**(2): 177-188.

Lopez-Herrejon, R. E., Ferrer, J., Chicano, F., Egyed, A., and Alba, E. 2016. *Evolutionary computation for software product line testing: An overview and open challenges*. Springer.

Mahmoud, T., and Ahmed, B. S. 2015. An efficient strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software testing use. *Expert Systems with Applications*. **42**(22): 8753-8765.

Mala, D. J., and Mohan, V. 2009. ABC tester-artificial bee colony based software test suite optimization approach. *International Journal of Software Engineering*. **2**(2): 15-43.

Malaiya, Y. K. 1995. Antirandom testing: getting the most out of black-box testing. *In the 6th International Symposium on Software Reliability Engineering*, 86-95.

Mandl, R. 1985. Orthogonal latin squares: an application of experiment design to compiler testing. *Communications of The ACM*. **28**(10): 1054-1058.

Mao, C., Yu, X., and Chen, J. 2012. Swarm intelligence-based test data generation for structural testing. *In the 11th International Conference on Computer and Information Science*, 623-628.

McCaffrey, J. D. 2009a. Generation of pairwise test sets using a genetic algorithm. *In the 33rd Annual IEEE International Computer Software and Applications Conference*, **1**, 626-631.

McCaffrey, J. D. 2009b. Generation of pairwise test sets using a simulated bee colony algorithm. *In the International Conference on Information Reuse and Integration*, 115-119.

McCaffrey, J. D. 2009c. Pairwise testing with QICT *Microsoft Developer Network Magazine* (Vol. 24, pp. 28-35): Microsoft.

McCaffrey, J. D. 2010. An empirical study of pairwise test set generation using a genetic algorithm. *In the 7th International Conference on Information Technology: New Generations*, 992-997.

McMinn, P. 2004. Search-based software test data generation: a survey. *Software Testing, Verification and Reliability*. **14**(2): 105-156.

Meng, X., Gao, X., and Liu, Y. 2015. A novel hybrid bat algorithm with differential evolution strategy for constrained optimization. *International Journal of Hybrid Information Technology*. **8**(1): 383-396.

Myers, G. J., Sandler, C., and Badgett, T. 2011. *The art of software testing*. 3rd ed. Hoboken: John Wiley and Sons.

Naik, K., and Tripathy, P. 2008. *Maturity models: Software testing and quality assurance: Theory and practice*. Citado John Wiley and Sons Online Library

Nasser, A., Alsariera, Y. A., Alsewari, A. A., and Zamli, K. Z. 2015. A cuckoo search based pairwise strategy for combinatorial testing problem. *Journal of Theoretical and Applied Information Technology*. **82**(1): 154-162.

Nasser, A. M., Alsariera, Y. A., Zamli, K. Z., and AlKazcmi, B. 2014. Late acceptance hill climbing based strategy for addressing constraints within combinatorial test data generation. *In the 7th Edition of Asia Software Testing Conference*, **7** (1).

Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B., and Alba, E. 2009. Mocell: a cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*. **24**(7): 726-746.

Nguyen, T. T., and Ho, S. D. 2016. Bat algorithm for economic emission load dispatch problem. *International Journal of Advanced Science and Technology*. **86**(1): 51-60.

Nie, C., and Leung, H. 2011. A survey of combinatorial testing. *ACM Computing Surveys*. **43**(2): 11.

Nursimulu, K., and Probert, R. L. 1995. Cause-effect graphing analysis and validation of requirements. *In the Centre for Advanced Studies on Collaborative Research Conference* 46.

Othman, R. R., Zamli, K. Z., and Syed Mohamad, S. M. 2013. t-way testing strategies: a critical survey and analysis. *International Journal of Digital Content Technology and Its Applications*. **7**(9): 222-235.

Pallas, D. 2003. Jenny test tool, Available from: http://www.burtleburtle.net./bob/math/jenny.html, last accessed on (April, 2016).

Pan, J. 1999. Software testing. *Dependable Embedded Systems, 5*(1), 1-14. Available from: http://www.ece.cmu.edu/~koopman/des_s99/sw_testing doi:10.1.1.103.7121

Pendharkar, P. C. 2010. Exhaustive and heuristic search approaches for learning a software defect prediction model. *Engineering Applications of Artificial Intelligence*. **23**(1): 34-40.

Petke, J. 2015, Constraints: The future of combinatorial interaction testing. *Paper presented at the* 8th International Workshop on Search-Based Software Testing.

Qi, R.-Z., Wang, Z.-J., and Li, S.-Y. 2016. A parallel genetic algorithm based on spark for pairwise test suite generation. *Journal of Computer Science and Technology*. **31**(2): 417-427.

Rabbi, K., Khatun, S., Yaakub, C. Y., and Klaib, M. 2011. EPS2Way: An efficient pairwise test data generation strategy. *International Journal of New Computer Architectures and Their Applications.* **1**(4): 1099-1109.

Rabbi, K., Mamun, Q., and Islam, M. R. 2015. An efficient particle swarm intelligence based strategy to generate optimum test data in t-way testing. *In the 10th Conference on Industrial Electronics and Applications*, 123-128.

Rabbi, K. F., Beg, A. H., and Herawan, T. 2012. *MT2Way: A novel strategy for pair-wise test data generation*. Springer.

Rahman, M., Othman, R. R., Ahmad, R. B., and Rahman, M. M. 2015. A meta heuristic search based t-way event driven input sequence test case generator. *International Journal of Simulation, Systems, Science and Technology.* **15**(3): 65-71.

Rakesh, H. V., Aruna, S. B., and Raju, T. D. 2013. Combined economic load and emission dispatch evalution using bat algorithm. *Indian Streams Research Journal.* **3**(5): 1-8.

Ramesh, B., Mohan, V. C. J., and Reddy, V. V. 2013. Application of bat algorithm for combined economic load and emission dispatch. *International Journal of Electricl Engineering and Telecommunications.* **2**(1): 1-9.

Rodrigues, D., Pereira, L. A. M., Nakamura, R. Y. M., Costa, K. A. P., Yang, X.-S., Souza, A. N., and Papa, J. P. 2014. A wrapper approach for feature selection based on bat algorithm and optimum-path forest. *Expert Systems with Applications.* **41**(5): 2250-2258.

Rodriguez-Cristerna, A., and Torres-Jimenez, J. 2012. A simulated annealing with variable neighborhood search approach to construct mixed covering arrays. *Electronic Notes in Discrete Mathematics.* **39**(1): 249-256.

Rodriguez-Cristerna, A., Torres-Jimenez, J., Gómez, W., and Pereira, W. C. A. 2015. Construction of mixed covering arrays using a combination of simulated annealing and variable neighborhood search. *Electronic Notes in Discrete Mathematics.* **47**(1): 109-116.

Roper, M. 2002. *Software testing*. 3rd ed. California: Encyclopedia of Physical Science and Technology Academic Press.

Sabharwal, S., Bansal, P., and Mittal, N. 2015. Construction of strength two mixed covering arrays using greedy mutation in genetic algorithm. *Information Technology and Computer Science.* **10**(1): 23-34.

Sabharwal, S., Bansal, P., and Mittal, N. 2017. Construction of t-way covering arrays using genetic algorithm. *International Journal of System Assurance Engineering And Management.* **8**(2): 264-274.

Sabharwal, S., Bansal, P., Mittal, N., and Malik, S. 2016. Construction of mixed covering arrays for pair-wise testing using probabilistic approach in genetic algorithm. *Arabian Journal for Science and Engineering.* **71**(8): 2821-2835.

Schroeder, P. J., Bolaki, P., and Gopu, V. 2004. Comparing the fault detection effectiveness of n-way and random test suites. *In the International Symposium on Empirical Software Engineering*, 49–59.

Senthilnath, J., Kulkarni, S., Benediktsson, J. A., and Yang, X.-S. 2016. A novel approach for multispectral satellite image classification based on the bat algorithm. *IEEE Geoscience and Remote Sensing Letters.* **13**(4): 599-603.

Sharma, M., and Chandra, S. 2010. Automatic generation of test suites from decision table - theory and implementation. *In the 5th International Conference on Software Engineering Advances*, 459-464.

Sherwood, G. 1994. Effective testing of factor combinations. *In the 3rd International Conference on Software Testing, Analysis and Review*.

Sherwood, G. 2003. TestCover, Available from: http://testcover.com/index.php, last accessed on (April, 2016).

Sherwood, G. B., Martirosyan, S. S., and Colbourn, C. J. 2005. Covering arrays of higher strength from permutation vectors. *Journal of Combinatorial Designs.* **14**(3): 202-213.

Shiba, T., Tsuchiya, T., and Kikuno, T. 2004. Using artificial life techniques to generate test cases for combinatorial testing. *In the 28th Annual International Conference on Computer Software and Applications*, 72-77.

Song, A., Li, M., Ding, X., Cao, W., and Pu, K. 2016. Community detection using discrete bat algorithm. *IAENG International Journal of Computer Science.* **43**(1): 37-43.

Srinivas, N., and Deb, K. 1994. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation.* **2**(3): 221-248.

Stardom, J. 2001. *Metaheuristics and the search for covering and packing array* Master. Thesis. Simon Fraser University, Canada.

Sthamer, H. H. 1995. *The automatic generation of software test data using genetic algorithms.* PhD. Thesis. University of Glamorgan, Pontyprid, Wales.

Sureja, N. M. 2012. New inspirations in nature: a survey. *International Journal of Computer Applications and Information Technology*. **1**(3): 21-24.

Taha, A. M., Mustapha, A., and Chen, S.-D. 2013. Naive bayes-guided bat algorithm for feature selection. *The Scientific World Journal.* **2013**(1): 1-9.

Tai, K. C., and Lei, Y. 2002. A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering* **28**(1): 109-111.

Timaná-Peña, J. A., Cobos-Lozada, C. A., and Torres-Jimenez, J. 2016. Metaheuristic algorithms for building covering arrays: A review. *Revista Facultad De Ingeniería.* **25**(43): 31-45.

Torres-Jimenez, J., and Rodriguez-Tello, E. 2012. New bounds for binary covering arrays using simulated annealing. *Information Sciences.* **185**(1): 137-152.

Tseng, C.-W., Mitra, S., Davidson, S., and McCluskey, E. J. 2001. An evaluation of pseudo random testing for detecting real defects. *In the 19th IEEE on VLSI Test Symposium*, 404-409.

Tung, Y.-W., and Aldiwan, W. S. 2000. Automating test case generation for the new generation mission software system. *In the Aerospace Conference* **1**, 431-437.

Wang, S., Ali, S., and Gotlieb, A. 2013. Minimizing test suites in software product lines using weight-based genetic algorithms. *In the 15th Annual Conference on Genetic and Evolutionary Computation*, 1493-1500.

Wang, Z., and He, H. 2013. Generating variable strength covering array for combinatorial software testing with greedy strategy. *Journal of Software*. **8**(12): 3173-3181.

Wang, Z. Y., Xu, B. W., and Nie, C. H. 2008. Greedy heuristic algorithms to generate variable strength combinatorial test suite. *In the 8th International Conference on Quality Software*, 155-160.

Wilcoxon, F. 1945. Individual comparisons by ranking methods. *Biometrics Bulletin*. **1**(6): 80-83.

Williams, A. W. 2000. Determination of test configurations for pair-wise interaction coverage. *In the 13th International Conference on the Testing of Communicating Systems*, 57-74.

Williams, A. W., Ho, J. H., and Lareau, A. 2003. TConfig test tool, Available from: http://www.site.uottawa.ca/~awilliam, School of Information Technology and Engineering (SITE), University of Ottawa, Ottawa, Ontario, Canada, last accessed on (April, 2016).

Williams, A. W., and Probert, R. L. 1996. A practical strategy for testing pair-wise coverage of network interfaces. *In the 7th International Symposium on Software Reliability Engineering*, 246-254.

Williams, A. W., and Probert, R. L. 2002. Formulation of the interaction test coverage problem as an integer program. *In the 14th International Conference on Testing of Communicating Systems*, 283-298.

Xiang, C., Qing, G., Ang, L., and Daoxu, C. 2009. Variable strength interaction testing with an ant colony system approach. *In the Asia-Pacific Software Engineering Conference*, 160-167.

Xiang, L. Y., Alsewari, A. A., and Zamli, K. Z. 2015. Pairwise test suite generator tool based on harmony search algorithm (HS-PTSGT). *NNGT International Journals on Artificial Intelligence*. **2**(1): 62-65.

Yang, X.-S. 2010. *A new metaheuristic bat-inspired algorithm*. Springer.

Yang, X.-S. 2014. *Nature-inspired optimization algorithms (chapter 10 – bat algorithms)*. 1st ed. Boston: Elsevier Science Publishers Inc.

Yang, X.-S., and Gandomi, A. H. 2012. Bat algorithm: A novel approach for global engineering optimization. *Engineering Computations.* **29**(5): 464-483.

Yilmaz, C., Cohen, M. B., and Porter, A. 2006. Covering arrays for efficient fault characterization in complex configuration spaces. *IEEE Transactions on Software Engineering.* **31**(1): 20–34.

Younis, M. I., and Zamli, K. Z. 2009a. ITTW: t-way minimization strategy based on intersection of tuples. *In the IEEE Symposium on Industrial Electronics and Applications*, **1**, 221-226.

Younis, M. I., and Zamli, K. Z. 2009b. RTS: Reverse tracking strategy for pairwise testing. *In the Conference on Software Engineering and Computer Systems*.

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

Younis, M. I., and Zamli, K. Z. 2010b. *MIPOG: A parallel t-way minimization strategy for combinatorial testing.* Ph.D. Thesis. Universiti Sains Malaysia, USM.

Younis, M. I., and 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.

Younis, M. I., Zamli, K. Z., and Isa, N. A. M. 2008a. Generating pairwise combinatorial test set using artificial parameters and values. *In the 3rd International Symposium on Information Technology*, **3**, 1654-1661.

Younis, M. I., Zamli, K. Z., and Isa, N. A. M. 2008b. IRPS: An efficient test data generation strategy for pairwise testing. *In the 12th International Conference on Knowledge-Based Intelligent Information and Engineering Systems*, 493-500.

Younis, M. I., Zamli, K. Z., Klaib, M. F. J., Soh, Z. H. C., Abdullah, S. A. C., and Isa, N. A. M. 2010. Assessing IRPS as an efficient pairwise test data generation strategy. *International Journal of Advanced Intelligence Paradigms.* **2**(3): 90-104.

Yu, L., Kacker, R., Kuhn, D. R., Okun, V., and Lawrence, J. 2008. IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing. *Software Testing Verification and Reliability.* **18**(3): 125-148.

Yu, L., Kacker, R., Kuhn, D. R., Okun, V., and Lawrence, J. 2009. IPOG/IPOD: Efficient test generation for multi-way software testing. *Journal of Software Testing, Verification, and Reliability.* **18**: 125 -148.

Yu, Y. T., Ng, S. P., and Chan, E. Y. K. 2003. Generating, selecting and prioritizing test cases from specifications with tool support. *In the 3rd International Conference on Quality Software*, 83-90.

Zabil, M. H. M., and Zamli, K. Z. 2013a. Adopting bees algorithm for sequence-based t-way test data generation. *In the International ICIC Express Letters*, **7**.

Zabil, M. H. M., and Zamli, K. Z. 2013b. Implementing a t-way test generation strategy using bees algorithm. *International Journal of Advances in Soft Computing and Its Applications.* **5**(3): 116-126.

Zabil, M. H. M., Zamli, K. Z., and Othman, R. 2012. Sequence-based interaction testing implementation using bees algorithm. *In the IEEE Symposium on Computers and Informatics*, 81-85.

Zamli, K. Z., Alkazemi, B. Y., and Kendall, G. 2016. A tabu search hyper-heuristic strategy for t-way test suite generation. *Applied Soft Computing.* **44**(1): 57-74.

Zamli, K. Z., Alsewari, A. R., and Al-Kazemi, B. 2015. Comparative benchmarking of constraints t-way test generation strategy based on late acceptance hill climbing algorithm. *International Journal of Software Engineering and Computer Systems.* **1**(1): 15-27.

Zamli, K. Z., Klaib, M. F., Younis, M. I., Isa, N. A. M., and 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.

Zekaoui, L. 2006. *Mixed covering arrays on graphs and tabu search algorithms.* Master. Thesis. University of Ottawa, Canada.

APPENDIX A
THE RUNNING COMMAND-LINE FOR BTS

BTS adapts special command-lines for advance used through a command prompt execution. These command-line has been designed for simplicity and to enable a faster process of the specify software inputs specifications. Additionally, the variables for BTS and operating environment as well.

Table A.1 The command-line specifications for BTS.

Command	Specifications
-d	-d <main-strength (t): Elements-values (ES)>
-m	-m <sub-strength (t): indexes of the involved **e**lements>
-r	-r <number of executions>
-n	-n <bat papulation size>
-i	-i <number of iteration (T_{max})>
-l	-l <loudness value>
-p	-l <emission of pulse rate value>
-t	-t <tolerance value>
-h	-h <Hamming distance limit>

Table A.2 Examples of command-line specifications for BTS.

Command	Specifications Example
-d	–d <2:5,5,5,5> The specification for 4 elements each has 5 values for 2-way.
-m	–m <3:1,2,3#3:2,3,4> The specification for the first three elements from the above-mentioned example with 3-way sub-strength and the last three for another 3-way sub-strength. Notice: all the indexes specified in BTS are started from 1 to N for simplicity and test engineers' convenience. We did not follows the default indexing method in programing, which starts from zero.
-r	-r <20> This command used for the sake of this research results to run the benchmarking for 20 times.
-n	-n <50>
-i	-i <100>
-l	-l <0.5>
-r	-r <0.25>
-t	-t <0.0001>
-h	-h <10>

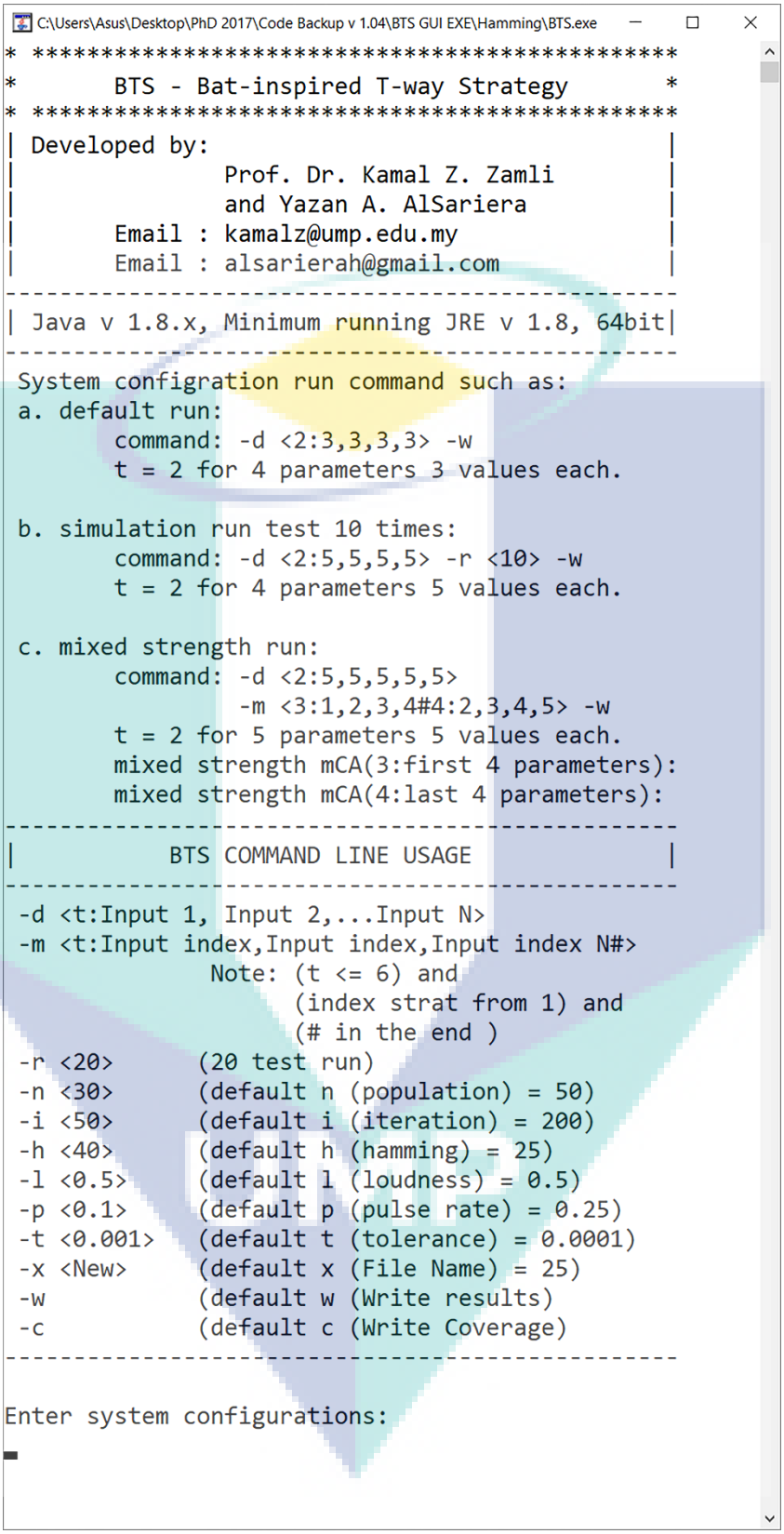


Figure A.1 The BTS advance user prototype.

APPENDIX B
BTS TUNING DATA

Table B.1 Full details of BTS tunning sizes and their averages.

Bat population size 10												
Loudness	**Pulse Rate**	**Tolerance**	**Iteration**									
			10		**20**		**50**		**100**		**200**	
			Size	*Average*	*Size*	*Average*	*Size*	*Average*	*Size*	*Average*	*Size*	*Average*
0.05	0.05	0.00001	38	39.85	37	37.95	35	36.85	35	36.40	35	36.10
		0.0001	37	39.30	37	37.90	36	36.90	36	36.85	34	36.10
		0.001	38	39.30	37	38.05	36	36.95	34	36.10	35	36.25
		0.01	38	40.00	36	37.70	35	36.80	35	36.30	34	35.80
		0.1	38	39.75	36	37.90	36	37.05	35	36.20	34	35.80
	0.25	0.00001	38	39.70	36	37.90	36	36.90	35	36.30	35	35.80
		0.0001	38	39.55	36	37.90	35	36.50	35	36.30	34	35.60
		0.001	38	39.70	36	37.70	36	37.00	35	36.60	35	36.25
		0.01	38	39.30	37	37.90	35	36.20	35	36.25	35	36.10
		0.1	38	39.55	36	37.75	35	36.65	35	36.40	35	35.90
	0.5	0.00001	38	39.20	36	37.90	35	36.70	35	36.20	35	36.15
		0.0001	38	39.80	35	37.50	35	36.50	35	36.25	34	35.80
		0.001	38	39.70	36	37.80	36	36.95	35	36.35	34	35.80
		0.01	37	39.30	37	37.70	35	36.60	35	36.45	35	36.15
		0.1	37	39.30	36	38.20	35	36.50	35	36.30	35	36.20
	0.75	0.00001	37	39.40	36	37.60	36	36.80	35	36.50	35	36.20
		0.0001	37	39.75	37	37.90	35	36.95	35	36.35	35	36.05
		0.001	38	39.75	36	37.90	35	36.50	35	36.40	35	36.10

		0.01	38	39.40	36	37.80	36	37.00	35	36.35	35	36.15
		0.1	36	39.00	36	38.00	36	36.75	35	36.60	35	36.35
	0.95	0.00001	38	39.45	35	37.75	34	36.90	35	36.05	35	35.95
		0.0001	38	39.55	36	37.65	35	36.70	35	36.60	35	36.20
		0.001	37	39.20	36	37.60	35	36.65	34	36.35	35	36.00
		0.01	38	39.40	36	38.10	36	37.00	35	36.35	34	36.15
		0.1	38	39.35	36	38.00	35	36.90	35	36.30	35	36.00
0.25	0.05	0.00001	38	39.55	36	37.95	35	36.55	35	36.55	35	36.25
		0.0001	38	39.40	37	38.10	35	36.65	35	36.45	35	36.15
		0.001	38	39.25	36	37.95	35	36.65	35	35.95	35	36.10
		0.01	37	39.50	36	37.95	36	36.85	34	36.10	35	36.10
		0.1	37	39.30	37	38.35	36	37.05	35	36.55	34	35.65
	0.25	0.00001	37	39.25	36	37.95	36	36.90	34	35.95	34	35.70
		0.0001	38	39.45	35	37.35	34	36.65	35	35.95	35	35.85
		0.001	38	39.25	37	38.15	35	36.90	35	36.25	35	36.00
		0.01	38	39.40	36	37.75	36	36.65	35	36.45	35	36.25
		0.1	38	39.80	36	37.70	36	36.90	34	36.10	34	36.15
	0.5	0.00001	37	39.35	36	37.70	35	36.85	34	36.10	35	36.10
		0.0001	38	39.30	37	38.10	35	36.65	35	36.40	35	35.95
		0.001	38	39.70	36	38.05	35	36.80	35	36.35	34	35.85
		0.01	38	39.75	36	37.70	35	36.75	35	36.35	35	36.25
		0.1	38	39.70	36	38.00	35	36.70	35	36.20	35	36.10
	0.75	0.00001	38	39.40	36	38.00	35	36.60	35	36.35	34	36.05
		0.0001	37	39.40	36	38.00	35	36.60	35	36.15	35	36.10
		0.001	37	39.40	35	37.60	34	36.70	35	36.30	35	36.05
		0.01	38	39.50	36	38.20	36	36.95	34	36.20	35	36.35
		0.1	38	39.70	36	37.85	35	36.75	35	36.40	34	35.95

	0.95	0.00001	38	39.50	35	37.85	36	36.55	35	36.45	34	35.65
		0.0001	38	39.65	36	38.10	35	36.75	34	36.30	35	35.80
		0.001	38	39.50	37	38.20	35	36.70	35	36.35	35	36.00
		0.01	38	39.35	37	38.05	35	36.70	34	35.90	35	35.90
		0.1	38	39.50	37	37.70	35	36.80	35	36.10	35	36.10
0.5	0.05	0.00001	38	39.40	37	38.30	36	36.80	35	36.10	35	36.05
		0.0001	38	39.85	36	37.85	35	36.75	35	36.45	35	36.30
		0.001	38	39.35	36	37.85	36	37.00	34	36.00	35	35.95
		0.01	38	39.70	37	38.10	36	36.60	35	36.40	35	35.95
		0.1	38	39.75	36	37.95	34	36.80	34	36.15	34	35.80
	0.25	0.00001	38	39.70	36	37.75	36	36.65	35	36.05	34	35.85
		0.0001	37	39.35	36	37.70	35	36.70	34	36.30	35	36.10
		0.001	38	39.55	36	38.10	34	36.80	35	36.30	35	36.05
		0.01	38	39.50	36	38.00	36	36.70	35	36.50	35	35.85
		0.1	39	39.65	36	37.85	35	36.75	35	36.75	35	36.25
	0.5	0.00001	38	39.65	36	37.80	35	36.60	35	36.40	34	35.75
		0.0001	38	39.65	36	37.75	36	36.75	35	36.05	34	35.55
		0.001	36	39.50	37	38.25	35	36.65	35	36.35	34	36.05
		0.01	38	39.30	36	38.00	35	37.05	35	36.40	35	36.15
		0.1	37	39.25	37	37.90	35	36.65	35	36.45	35	36.00
	0.75	0.00001	38	39.65	36	37.65	36	36.55	35	36.30	35	36.00
		0.0001	38	39.55	36	37.85	36	37.05	35	36.20	35	36.10
		0.001	38	39.30	37	37.95	36	37.05	35	36.30	35	36.10
		0.01	38	39.30	36	37.80	36	37.20	34	36.10	33	35.75
		0.1	38	39.00	37	37.85	35	36.85	35	36.30	35	36.15
	0.95	0.00001	38	39.60	37	37.80	35	36.90	35	36.55	34	36.00
		0.0001	37	39.55	36	37.95	36	37.15	34	36.50	35	36.10

		0.001	38	39.60	37	38.05	35	37.15	35	36.40	35	36.25
		0.01	37	39.50	37	37.95	36	36.85	34	35.90	35	36.35
		0.1	38	39.40	36	37.75	36	36.70	34	36.05	35	36.00
0.75	0.05	0.00001	38	40.00	37	37.85	35	36.95	35	36.15	35	35.85
		0.0001	38	39.05	36	37.75	36	36.80	35	36.15	35	36.15
		0.001	37	39.20	37	38.00	35	36.50	35	36.35	35	36.10
		0.01	38	39.45	36	37.80	35	36.90	35	36.25	34	35.90
		0.1	38	39.35	36	38.20	36	36.95	35	36.55	35	36.15
	0.25	0.00001	38	39.70	36	37.80	35	36.85	35	36.25	35	35.95
		0.0001	38	39.40	36	37.75	35	36.45	35	36.10	35	36.25
		0.001	37	39.65	37	37.80	36	36.95	35	36.65	35	36.05
		0.01	38	39.20	36	37.75	35	36.60	35	36.10	35	35.90
		0.1	38	39.80	36	37.70	35	36.65	35	36.20	36	36.30
	0.5	0.00001	38	39.45	36	37.65	36	36.55	36	36.60	35	36.05
		0.0001	37	39.95	36	37.80	35	36.85	35	36.00	34	35.55
		0.001	38	39.20	37	37.75	35	36.80	35	36.10	35	36.15
		0.01	38	39.60	37	38.15	35	37.00	35	36.25	35	36.05
		0.1	38	39.60	36	37.90	35	36.55	35	36.35	35	36.05
	0.75	0.00001	38	39.40	36	37.80	36	36.85	35	36.40	35	36.05
		0.0001	38	39.60	37	38.10	35	36.35	35	36.15	34	35.90
		0.001	38	39.40	37	38.25	36	36.90	35	36.15	35	36.00
		0.01	38	39.50	37	38.15	36	36.65	35	36.65	34	35.80
		0.1	38	39.80	37	38.20	36	36.60	34	36.05	34	35.95
	0.95	0.00001	37	39.40	35	38.00	35	36.35	34	36.45	34	35.95
		0.0001	38	39.05	36	37.75	35	36.75	35	36.10	35	36.20
		0.001	38	39.30	36	37.55	35	36.75	35	36.20	34	35.75
		0.01	37	39.75	37	37.75	35	36.65	35	36.20	34	36.05

		0.1	38	39.65	36	37.85	35	36.65	35	36.10	34	35.75
0.95	0.05	0.00001	38	39.50	36	37.60	36	36.85	34	36.15	35	36.20
		0.0001	38	39.30	36	37.55	36	36.60	35	36.30	35	36.00
		0.001	38	39.60	36	37.85	36	36.90	35	36.55	35	35.75
		0.01	38	39.60	37	37.85	36	36.85	34	36.30	34	35.75
		0.1	37	39.15	37	37.85	35	36.70	35	36.45	35	35.90
	0.25	0.00001	37	39.15	36	37.95	35	36.95	34	35.95	35	35.90
		0.0001	37	39.35	37	37.90	36	36.75	35	36.35	35	36.15
		0.001	37	39.40	36	37.55	36	37.05	35	36.20	33	35.70
		0.01	36	38.95	36	38.00	35	36.55	35	36.25	35	36.05
		0.1	38	39.55	36	37.75	35	36.50	35	36.20	35	36.05
	0.5	0.00001	38	39.85	36	37.80	35	36.90	35	36.35	35	36.40
		0.0001	37	39.35	36	38.00	35	36.75	35	36.65	35	35.75
		0.001	38	39.45	37	37.80	36	37.05	35	36.30	33	36.00
		0.01	38	39.35	36	37.50	35	36.70	35	36.20	35	35.95
		0.1	38	39.20	36	38.05	35	37.00	34	36.20	35	36.25
	0.75	0.00001	38	39.10	36	37.65	36	36.85	35	35.90	35	35.85
		0.0001	37	39.25	37	38.30	36	36.85	35	36.40	35	36.00
		0.001	37	39.60	36	37.90	36	36.75	34	36.15	35	36.00
		0.01	37	39.65	36	37.90	35	36.70	35	36.25	35	36.15
		0.1	38	39.65	36	37.70	35	36.90	34	36.10	34	35.85
	0.95	0.00001	38	39.60	37	37.65	36	37.00	35	36.35	35	36.40
		0.0001	37	39.35	37	37.80	36	37.05	35	36.70	34	35.80
		0.001	36	38.90	38	38.45	35	36.50	35	36.30	35	36.00
		0.01	38	39.30	36	38.00	35	36.80	34	36.00	35	35.90
		0.1	38	39.65	36	37.80	36	36.80	35	36.15	35	36.20

Bat population size 20												
0.05	0.05	0.00001	37	37.75	36	37.05	35	36.20	34	35.95	34	35.60
		0.0001	37	38.00	36	36.80	35	36.20	34	35.80	34	35.85
		0.001	36	37.55	35	36.50	35	36.20	34	35.55	34	35.45
		0.01	37	38.10	35	36.75	35	35.95	33	35.60	34	35.60
		0.1	36	37.65	35	36.85	36	36.70	33	35.70	34	35.75
	0.25	0.00001	36	37.70	36	36.75	35	35.85	35	36.05	35	35.90
		0.0001	37	38.10	36	36.90	34	36.25	35	35.90	34	35.80
		0.001	36	37.85	35	37.10	35	36.05	35	35.95	33	35.55
		0.01	37	37.75	36	37.00	35	35.80	35	35.95	35	35.60
		0.1	36	37.60	35	36.65	34	36.00	35	35.70	35	35.80
	0.5	0.00001	36	37.75	36	36.65	35	35.90	34	35.75	34	35.65
		0.0001	37	38.00	35	36.55	35	36.40	35	35.95	34	35.70
		0.001	36	37.60	36	36.95	34	35.90	34	35.60	34	35.50
		0.01	36	37.45	35	36.50	34	36.05	34	36.15	34	35.75
		0.1	36	37.55	35	36.55	35	36.00	34	35.70	34	35.65
	0.75	0.00001	36	37.80	36	36.70	34	35.85	34	36.15	33	35.60
		0.0001	36	37.50	35	36.60	35	36.25	35	36.10	34	35.40
		0.001	36	37.90	35	36.75	35	36.65	35	36.05	34	35.65
		0.01	37	37.95	35	36.85	35	36.25	34	35.90	35	35.90
		0.1	36	37.55	35	36.35	35	35.75	34	35.70	34	35.75
	0.95	0.00001	36	38.10	35	36.70	35	36.10	34	35.75	34	35.75
		0.0001	36	37.70	36	36.85	35	36.10	34	35.60	34	35.70
		0.001	37	37.80	35	36.55	35	36.30	34	35.80	34	35.90
		0.01	36	38.05	35	36.70	35	36.15	34	35.70	34	35.50
		0.1	36	37.80	35	37.00	34	35.80	34	35.60	35	35.65
0.25	0.05	0.00001	35	37.45	35	36.65	35	36.00	34	35.55	34	35.75

		0.0001	36	37.40	36	36.90	35	36.10	34	35.95	34	35.35
		0.001	37	38.10	35	36.50	34	36.15	35	36.00	35	35.85
		0.01	36	38.00	35	36.90	34	36.00	35	35.50	34	35.75
		0.1	36	37.75	34	37.05	35	36.05	34	35.95	34	35.45
	0.25	0.00001	36	37.75	35	36.65	34	36.00	35	35.75	32	35.55
		0.0001	36	37.90	35	36.65	35	36.15	35	35.95	33	35.75
		0.001	36	37.40	35	37.05	35	36.45	35	36.05	34	35.40
		0.01	36	37.65	35	36.60	33	36.00	35	36.10	34	35.70
		0.1	36	37.90	36	36.90	35	35.75	34	35.80	34	35.75
	0.5	0.00001	36	37.60	35	36.95	35	35.95	35	36.00	34	35.70
		0.0001	37	37.95	36	37.05	35	36.20	34	35.70	35	36.05
		0.001	36	37.60	35	36.70	35	35.95	35	35.50	34	35.45
		0.01	36	37.80	35	36.85	34	35.90	34	35.95	35	35.65
		0.1	36	37.55	35	36.60	35	36.20	33	35.60	35	35.80
	0.75	0.00001	36	37.55	36	36.80	35	36.25	34	35.80	34	35.30
		0.0001	36	37.75	35	36.65	35	35.90	34	35.65	34	35.70
		0.001	36	37.75	35	36.75	34	35.95	35	35.85	34	35.75
		0.01	35	37.65	35	36.80	35	36.20	35	36.00	34	35.25
		0.1	36	37.70	35	36.80	34	35.80	34	36.05	33	35.45
	0.95	0.00001	37	38.10	34	36.55	35	36.35	34	35.95	35	36.00
		0.0001	36	37.95	35	36.65	35	35.90	34	35.85	34	35.40
		0.001	36	37.55	35	36.75	35	35.95	34	35.65	34	35.45
		0.01	36	37.70	35	36.50	34	36.10	34	35.85	34	35.70
		0.1	36	37.90	35	36.75	34	35.60	35	36.00	35	35.45
0.5	0.05	0.00001	36	37.70	35	36.55	34	36.00	35	35.85	34	35.40
		0.0001	36	37.80	35	36.75	35	36.10	33	35.55	34	35.60
		0.001	36	37.65	36	36.60	34	36.10	34	35.80	33	35.65

		0.01	37	38.35	35	36.95	35	36.15	35	36.00	33	35.70
		0.1	36	37.50	36	36.90	35	36.15	34	35.75	35	35.60
	0.25	0.00001	35	37.80	35	36.70	35	35.90	35	35.75	34	35.35
		0.0001	36	37.55	35	36.60	34	36.00	34	36.10	34	35.60
		0.001	36	37.70	35	36.65	35	35.90	34	35.50	34	35.70
		0.01	37	38.25	36	37.05	35	36.25	35	36.05	34	35.50
		0.1	36	37.90	36	36.60	34	35.65	34	35.80	34	35.30
	0.5	0.00001	36	37.70	35	36.75	35	36.10	33	35.50	35	35.85
		0.0001	36	37.45	35	36.40	35	36.00	34	35.65	33	35.65
		0.001	36	37.45	35	36.65	34	36.05	34	35.65	34	35.45
		0.01	37	38.05	35	36.90	35	36.10	35	35.75	35	35.85
		0.1	36	37.75	35	36.85	35	36.25	35	35.70	34	35.75
	0.75	0.00001	37	37.90	36	36.80	35	36.15	35	35.85	35	35.85
		0.0001	37	37.80	36	36.60	35	35.80	34	35.70	33	35.55
		0.001	37	37.80	35	36.35	35	36.00	35	35.85	34	35.55
		0.01	36	37.95	35	36.40	35	36.20	34	35.60	34	35.75
		0.1	37	37.70	35	36.90	35	35.85	35	36.00	35	36.00
	0.95	0.00001	37	38.20	35	36.60	35	36.35	34	35.55	34	35.50
		0.0001	36	37.65	35	36.60	35	36.35	34	35.70	34	35.45
		0.001	36	37.80	35	36.75	34	35.95	34	35.65	34	35.25
		0.01	36	37.80	36	36.90	34	35.80	35	35.65	34	35.70
		0.1	36	37.75	35	36.70	34	36.05	33	35.40	34	35.95
0.75	0.05	0.00001	36	37.65	35	36.70	34	35.80	34	35.75	34	35.25
		0.0001	37	37.85	35	36.75	35	35.85	35	35.95	34	35.55
		0.001	37	37.80	36	36.60	35	35.90	33	36.10	34	35.65
		0.01	36	37.85	35	36.75	35	36.05	34	35.70	34	35.70
		0.1	37	37.65	35	36.60	34	36.00	34	35.75	33	35.50

	0.25	0.00001	37	38.00	35	36.45	35	36.05	33	35.80	35	35.90
		0.0001	36	37.95	35	36.70	35	35.75	34	35.90	34	35.60
		0.001	37	37.90	36	36.90	35	36.00	35	35.70	35	35.80
		0.01	36	37.60	35	36.35	34	36.00	34	35.85	35	35.65
		0.1	36	37.70	36	36.85	35	36.35	34	36.00	33	35.45
	0.5	0.00001	36	37.60	35	36.65	34	36.20	35	35.80	34	35.90
		0.0001	36	37.90	35	36.55	35	36.20	34	35.95	34	35.95
		0.001	37	37.65	35	36.50	35	36.15	35	35.65	34	35.65
		0.01	37	37.90	35	36.60	34	36.20	33	35.95	34	35.70
		0.1	36	38.15	36	37.10	35	36.35	34	35.55	34	35.65
	0.75	0.00001	36	37.70	36	36.80	34	36.10	34	35.80	35	35.70
		0.0001	36	37.60	36	36.80	35	36.00	35	35.70	34	35.75
		0.001	36	37.80	35	37.05	32	35.80	35	35.75	34	35.55
		0.01	37	37.70	35	36.80	35	35.90	33	35.90	35	35.80
		0.1	36	37.65	34	36.25	35	36.25	35	35.65	33	35.30
	0.95	0.00001	36	38.05	36	36.65	35	36.40	34	35.75	33	35.60
		0.0001	36	37.40	35	36.55	34	36.00	34	35.30	34	35.60
		0.001	36	38.00	35	36.70	34	36.00	35	35.50	34	35.40
		0.01	36	37.30	35	36.85	33	35.80	35	35.65	34	35.65
		0.1	36	37.55	35	36.45	35	36.10	35	35.90	34	35.80
0.95	0.05	0.00001	36	37.60	35	36.60	34	36.20	34	35.95	35	35.55
		0.0001	36	37.70	36	36.80	35	36.25	34	35.45	33	35.50
		0.001	35	37.45	35	36.80	35	36.05	33	35.70	35	35.60
		0.01	37	37.80	35	36.45	33	35.95	34	35.75	34	35.40
		0.1	36	37.55	35	36.65	35	36.10	34	36.10	34	35.75
	0.25	0.00001	36	37.80	35	36.55	35	36.20	35	36.00	34	35.70
		0.0001	36	37.70	36	37.00	35	36.20	34	35.65	33	35.30

		0.001	36	37.70	35	36.40	35	36.00	35	36.10	35	35.75
		0.01	36	37.75	35	36.45	35	36.15	34	35.75	34	35.55
		0.1	35	38.00	36	36.65	34	36.05	35	35.85	33	35.05
	0.5	0.00001	36	37.60	34	36.45	33	35.70	33	35.80	34	35.65
		0.0001	37	37.90	35	36.85	34	35.90	35	35.80	34	35.75
		0.001	36	37.55	35	36.80	35	35.95	34	35.90	34	35.70
		0.01	37	37.85	34	36.70	35	36.00	35	35.75	35	36.00
		0.1	35	37.65	35	36.50	35	36.15	34	35.75	34	35.65
	0.75	0.00001	37	37.70	35	36.60	34	35.95	34	35.85	35	35.65
		0.0001	36	37.55	35	37.00	35	35.95	34	35.85	33	35.65
		0.001	36	37.55	36	36.85	35	36.00	34	35.90	34	35.90
		0.01	36	37.65	35	36.40	34	35.80	34	35.70	35	35.80
		0.1	36	37.95	35	36.50	35	36.10	34	35.75	34	35.35
	0.95	0.00001	36	37.65	36	36.90	34	36.00	34	35.55	34	35.75
		0.0001	36	37.40	35	36.45	35	36.05	35	35.85	34	35.65
		0.001	37	37.85	35	36.90	35	35.90	35	35.90	32	35.45
		0.01	36	37.60	35	36.50	35	36.05	34	35.60	33	35.60
		0.1	35	37.45	35	36.65	35	36.00	35	35.80	34	35.25
Bat population size 50												
0.05	0.05	0.00001	34	36.35	35	35.85	34	35.55	34	35.65	33	35.15
		0.0001	35	36.30	35	35.75	33	35.45	34	35.50	34	35.35
		0.001	34	36.55	34	35.75	34	35.85	34	35.60	33	35.40
		0.01	35	36.55	35	35.75	34	35.85	34	35.50	33	35.15
		0.1	35	36.30	34	35.80	34	35.60	35	35.60	34	35.35
	0.25	0.00001	35	36.35	35	36.00	34	35.75	34	35.95	35	35.90
		0.0001	35	36.50	34	35.60	35	35.90	34	35.40	34	35.40
		0.001	35	36.45	34	35.70	34	35.70	34	35.55	34	35.20

		0.01	35	36.25	35	36.10	35	35.85	33	35.35	33	35.50
		0.1	35	35.95	34	35.90	34	35.40	35	35.55	34	35.40
	0.5	0.00001	35	36.55	35	36.15	34	35.70	33	35.80	34	35.45
		0.0001	35	36.25	35	35.95	34	35.65	34	35.45	34	35.15
		0.001	35	36.15	34	36.00	35	35.90	35	35.70	34	35.35
		0.01	35	36.55	34	35.80	34	35.35	33	35.40	34	35.50
		0.1	35	36.00	35	35.65	34	35.85	34	35.60	34	35.35
	0.75	0.00001	35	36.60	35	36.05	34	35.35	34	35.50	34	35.35
		0.0001	34	36.15	35	35.95	34	35.55	34	35.75	34	35.55
		0.001	35	36.35	34	35.60	34	35.50	34	35.40	34	35.35
		0.01	35	36.35	34	35.80	34	35.80	34	35.20	34	35.50
		0.1	35	36.40	35	35.85	33	35.60	33	35.45	35	35.75
	0.95	0.00001	35	36.55	35	36.20	34	35.70	34	35.30	34	35.35
		0.0001	35	36.30	35	35.95	34	35.85	34	35.55	33	35.40
		0.001	35	36.55	35	35.90	35	35.55	34	35.55	34	35.40
		0.01	34	36.55	34	36.05	34	35.55	34	35.60	35	35.70
		0.1	35	36.25	33	35.50	34	35.25	33	35.40	34	35.55
0.25	0.05	0.00001	35	36.30	34	35.55	34	35.55	33	35.55	34	35.35
		0.0001	35	36.60	34	35.50	34	35.70	34	35.45	34	35.20
		0.001	35	36.20	35	36.05	34	35.65	34	35.65	33	34.75
		0.01	35	36.50	34	35.65	34	35.55	35	35.70	34	35.45
		0.1	35	36.35	34	35.75	35	35.50	34	35.55	34	35.55
	0.25	0.00001	35	36.40	35	36.15	35	35.90	33	35.50	34	35.55
		0.0001	35	36.35	35	35.75	34	35.55	33	35.40	32	35.25
		0.001	35	36.20	35	35.95	33	35.45	34	35.45	34	35.35
		0.01	34	36.45	35	36.10	34	35.90	34	35.50	34	35.40
		0.1	35	36.45	35	36.25	35	35.80	34	35.65	35	35.70

	0.5	0.00001	35	36.05	34	35.75	34	35.60	34	35.55	34	35.55
		0.0001	35	36.35	35	36.05	35	35.90	34	35.50	34	35.45
		0.001	35	36.40	34	35.70	35	35.85	34	35.45	34	35.10
		0.01	35	36.35	35	35.70	34	35.35	35	35.55	34	35.70
		0.1	35	36.30	35	35.60	34	35.45	33	35.35	34	35.35
	0.75	0.00001	34	36.10	35	35.85	34	35.35	33	35.30	34	35.45
		0.0001	35	36.25	35	36.05	35	36.10	34	35.35	33	35.50
		0.001	35	36.45	34	36.05	34	35.50	34	35.55	34	35.40
		0.01	35	36.20	35	35.50	34	35.35	34	35.40	34	35.65
		0.1	35	36.20	34	35.90	32	35.15	35	35.50	34	35.40
	0.95	0.00001	35	36.35	34	35.80	34	35.70	34	35.65	34	35.50
		0.0001	35	36.10	34	35.55	35	35.55	34	35.80	34	34.95
		0.001	33	36.25	34	36.15	34	35.95	34	35.70	33	35.40
		0.01	35	36.15	35	35.90	35	35.75	33	34.90	34	35.60
		0.1	35	36.30	34	35.90	34	35.75	34	35.20	33	35.20
0.5	0.05	0.00001	35	36.35	34	35.60	33	35.45	33	35.50	34	35.40
		0.0001	35	36.35	35	36.05	34	35.55	35	35.80	34	35.35
		0.001	34	36.45	35	36.20	34	35.65	33	35.45	34	35.80
		0.01	35	35.85	35	36.05	34	35.85	33	35.20	35	35.80
		0.1	35	36.50	35	35.75	35	35.55	33	35.55	33	35.40
	0.25	0.00001	36	36.65	34	35.90	34	35.70	34	35.70	34	35.45
		0.0001	35	36.45	35	35.85	34	35.50	34	35.75	34	35.55
		0.001	35	36.45	35	35.85	34	35.60	34	35.30	34	35.45
		0.01	34	36.20	35	35.70	35	35.55	34	35.20	34	35.15
		0.1	35	36.40	34	35.80	34	35.65	34	35.25	34	35.55
	0.5	0.00001	34	36.15	34	35.75	34	35.65	34	35.45	33	35.40
		0.0001	34	36.00	34	35.70	34	35.70	34	35.70	33	35.35

		0.001	35	36.35	35	35.85	33	35.60	34	35.65	34	35.45
		0.01	35	36.50	35	35.70	34	35.60	35	35.80	34	35.20
		0.1	35	36.20	35	36.05	33	35.65	34	35.55	34	35.15
	0.75	0.00001	35	36.35	34	36.20	34	35.55	34	35.70	34	35.45
		0.0001	35	36.50	34	35.55	34	35.55	33	35.45	34	35.30
		0.001	35	36.35	34	35.80	34	35.70	34	35.25	33	35.25
		0.01	36	36.30	35	35.85	34	35.45	34	35.30	33	35.25
		0.1	35	36.25	35	36.15	34	35.45	34	35.30	34	35.40
	0.95	0.00001	35	36.40	35	35.95	34	35.75	34	35.75	34	35.40
		0.0001	35	36.45	34	36.15	35	36.00	34	35.60	34	35.50
		0.001	35	36.10	35	35.85	34	35.45	35	35.70	34	35.65
		0.01	35	36.30	34	35.85	34	35.55	35	35.60	34	35.45
		0.1	36	36.50	35	35.55	34	35.55	35	35.75	34	35.75
0.75	0.05	0.00001	35	36.65	34	35.85	34	35.50	33	35.25	33	35.55
		0.0001	35	36.50	35	35.70	34	35.60	33	35.40	34	35.35
		0.001	35	36.55	35	35.70	34	35.65	34	35.60	34	35.45
		0.01	35	36.40	34	35.65	34	35.55	34	35.50	34	35.25
		0.1	35	36.35	33	35.75	34	35.65	34	35.45	34	35.30
	0.25	0.00001	35	36.45	34	35.75	34	35.70	33	35.20	34	35.40
		0.0001	35	36.55	35	36.15	33	35.40	34	35.25	33	35.25
		0.001	35	36.50	35	35.85	33	35.60	34	35.30	34	35.35
		0.01	35	36.20	34	35.65	34	35.55	35	35.60	34	35.45
		0.1	35	36.15	35	36.10	34	35.65	34	35.30	34	35.40
	0.5	0.00001	35	36.55	34	35.70	35	35.60	34	35.60	34	35.30
		0.0001	36	36.30	35	36.10	35	35.70	34	35.75	33	35.40
		0.001	35	36.15	34	35.50	34	35.70	34	35.30	34	35.35
		0.01	34	36.30	35	35.95	34	35.45	33	35.55	34	34.95

		0.1	35	36.25	34	35.80	33	35.00	34	35.45	34	35.40
	0.75	0.00001	34	36.25	35	36.15	34	35.55	35	35.60	33	35.10
		0.0001	35	36.05	33	35.80	34	35.60	34	35.45	34	35.35
		0.001	35	36.35	34	35.90	34	35.55	34	35.25	34	35.25
		0.01	34	36.40	34	35.85	33	35.70	34	35.50	34	35.35
		0.1	35	36.45	35	36.05	34	35.35	34	35.80	34	35.75
	0.95	0.00001	35	36.05	34	35.95	34	35.35	33	35.45	34	35.60
		0.0001	34	36.40	34	36.15	35	35.90	34	35.75	34	35.65
		0.001	35	36.35	34	35.75	34	35.80	34	35.10	33	35.00
		0.01	34	36.45	35	35.80	34	35.45	34	35.45	34	35.40
		0.1	35	36.25	35	36.10	33	35.10	33	35.50	34	35.45
0.95	0.05	0.00001	35	36.35	35	35.65	34	35.45	34	35.15	34	35.60
		0.0001	34	36.45	34	35.65	34	35.40	35	35.55	34	35.30
		0.001	35	36.20	34	35.75	34	35.30	33	35.70	34	35.35
		0.01	35	36.35	35	35.80	35	35.50	35	35.65	34	35.70
		0.1	34	36.15	34	35.70	34	35.30	33	35.25	34	35.25
	0.25	0.00001	35	36.45	35	36.05	33	35.70	34	35.45	34	35.65
		0.0001	35	36.35	34	35.75	34	35.40	34	35.25	35	35.35
		0.001	35	36.05	35	35.35	34	35.60	35	35.55	34	35.35
		0.01	34	36.00	34	35.90	34	35.25	34	35.45	34	35.35
		0.1	35	36.40	34	35.95	34	35.55	33	35.15	34	35.45
	0.5	0.00001	34	36.15	34	35.55	34	35.55	33	35.45	34	35.35
		0.0001	35	36.50	34	35.85	34	35.55	34	35.65	34	35.65
		0.001	34	36.05	34	35.60	34	35.55	34	35.60	34	35.60
		0.01	35	36.15	35	35.70	33	35.45	35	35.65	33	35.05
		0.1	35	36.15	34	35.85	34	35.45	34	35.70	34	35.30
	0.75	0.00001	35	36.20	34	35.80	33	35.35	34	35.05	33	35.35

		0.0001	35	35.90	34	35.75	35	35.85	33	35.15	34	35.45
		0.001	35	36.25	35	35.90	35	35.90	34	35.30	34	35.65
		0.01	35	36.35	35	35.70	34	35.65	34	35.45	34	35.45
		0.1	34	36.10	34	35.80	35	35.60	34	35.30	33	35.25
	0.95	0.00001	35	36.30	35	36.20	34	35.20	34	35.60	35	35.75
		0.0001	35	36.40	35	36.00	34	35.85	34	35.30	34	35.20
		0.001	34	36.70	34	35.80	35	35.60	35	35.65	33	35.20
		0.01	35	36.50	33	35.70	35	35.75	34	35.20	34	35.80
		0.1	35	36.40	34	35.90	33	35.55	34	35.45	34	35.55
Bat population size 100												
0.05	0.05	0.00001	34	35.30	34	35.45	33	35.35	34	35.15	34	35.10
		0.0001	35	36.00	34	35.30	34	35.25	34	35.35	33	35.35
		0.001	35	36.20	34	35.70	34	35.55	34	35.40	33	35.15
		0.01	35	35.95	34	35.70	34	35.30	34	35.55	34	35.35
		0.1	34	36.15	34	35.50	34	35.40	34	35.30	33	35.15
	0.25	0.00001	34	35.75	34	35.45	34	35.55	34	35.45	34	35.45
		0.0001	35	35.75	34	35.40	35	35.45	34	35.65	35	35.65
		0.001	34	35.70	35	35.75	33	35.00	34	35.85	32	35.30
		0.01	35	35.75	33	35.45	34	35.45	34	35.50	34	35.45
		0.1	34	36.00	34	35.40	34	35.25	34	34.90	33	35.30
	0.5	0.00001	34	35.90	34	35.45	33	35.30	34	35.35	34	35.45
		0.0001	35	35.80	34	35.50	35	35.50	35	35.80	34	35.25
		0.001	34	35.30	34	35.50	34	35.20	33	34.95	34	35.25
		0.01	34	35.60	35	35.80	34	35.50	35	35.80	34	35.45
		0.1	34	36.00	34	35.30	35	35.50	34	35.35	34	35.35
	0.75	0.00001	35	35.75	34	35.35	34	35.45	34	35.75	33	35.25
		0.0001	34	35.75	34	35.75	34	35.90	34	35.40	33	35.25

		0.001	35	35.90	34	35.50	34	35.35	33	35.35	34	35.35
		0.01	35	35.75	35	35.75	34	35.45	34	35.45	34	35.30
		0.1	34	35.75	34	35.45	34	35.35	34	35.05	34	35.55
	0.95	0.00001	34	35.80	34	35.35	34	35.45	34	35.40	34	35.30
		0.0001	34	35.85	34	35.75	34	35.30	33	35.35	34	35.25
		0.001	34	35.65	34	35.70	34	35.35	34	35.50	33	35.30
		0.01	35	35.90	34	35.60	34	35.45	34	35.55	34	35.35
		0.1	35	35.90	34	35.65	33	35.30	34	35.60	34	35.60
0.25	0.05	0.00001	34	35.75	34	35.65	33	35.10	34	35.15	34	35.40
		0.0001	34	35.90	34	35.75	34	35.60	33	35.25	33	35.10
		0.001	34	35.60	35	35.80	34	35.40	33	35.00	34	35.40
		0.01	34	35.65	34	35.65	35	35.85	34	35.45	33	35.35
		0.1	35	36.10	35	35.75	34	35.45	33	35.00	33	35.10
	0.25	0.00001	34	35.75	34	35.80	34	35.60	34	35.70	34	35.50
		0.0001	34	35.95	33	35.25	34	35.75	33	35.20	33	35.55
		0.001	34	35.50	33	35.25	34	35.30	34	35.60	34	35.60
		0.01	34	35.95	34	35.70	34	35.70	34	35.45	33	35.30
		0.1	35	35.70	35	35.65	34	35.30	34	35.30	33	35.00
	0.5	0.00001	35	35.55	35	35.55	33	35.50	35	35.55	35	35.70
		0.0001	35	35.95	34	35.85	34	35.05	33	35.25	34	35.35
		0.001	35	36.00	33	35.50	34	35.45	34	35.70	34	35.40
		0.01	34	35.55	34	35.60	34	35.45	34	35.40	34	35.55
		0.1	34	35.80	34	35.75	33	35.20	34	35.35	33	34.95
	0.75	0.00001	35	36.10	35	35.90	35	35.65	33	35.35	34	35.40
		0.0001	34	35.70	34	35.30	34	35.35	33	35.50	35	35.45
		0.001	35	35.90	34	35.15	34	35.60	34	35.25	33	35.35
		0.01	34	35.60	35	35.55	34	35.35	35	35.50	34	35.45

		0.1	34	35.80	34	35.80	33	35.20	35	35.60	34	35.25
	0.95	0.00001	34	35.60	34	35.80	35	35.85	34	35.40	33	35.30
		0.0001	34	35.85	34	35.65	34	35.15	35	36.00	34	35.10
		0.001	33	35.65	32	35.15	34	35.30	33	35.50	33	35.40
		0.01	34	35.80	35	35.70	33	35.05	34	35.45	34	35.00
		0.1	35	35.55	35	35.70	34	35.40	34	35.60	33	35.60
0.5	0.05	0.00001	34	35.65	34	35.75	34	35.25	34	35.40	34	35.20
		0.0001	35	36.00	34	35.35	34	35.15	33	35.20	34	35.50
		0.001	34	35.80	34	35.35	34	35.70	34	35.45	34	35.55
		0.01	35	35.80	33	35.15	34	35.50	34	35.55	34	35.25
		0.1	35	35.75	34	35.60	33	35.45	34	35.65	34	35.25
	0.25	0.00001	34	35.80	34	35.50	34	35.55	34	35.30	33	35.20
		0.0001	35	36.05	34	35.35	34	35.35	34	35.40	34	35.15
		0.001	33	35.75	34	35.45	35	35.55	34	34.90	34	35.80
		0.01	35	36.10	34	35.40	34	35.75	34	35.35	33	35.30
		0.1	35	35.85	34	35.30	35	35.45	34	35.40	33	35.10
	0.5	0.00001	34	35.60	34	35.80	35	35.40	34	35.30	34	35.10
		0.0001	35	35.80	35	35.40	34	35.50	35	35.50	35	35.60
		0.001	34	35.95	33	35.45	34	35.15	34	35.35	34	35.25
		0.01	35	36.10	34	35.35	34	35.75	33	35.25	34	35.70
		0.1	35	35.90	35	35.50	35	35.65	33	34.90	33	35.15
	0.75	0.00001	34	35.80	34	35.55	33	35.50	34	35.55	34	35.45
		0.0001	35	36.05	34	36.10	34	35.65	34	35.10	35	35.75
		0.001	34	35.80	33	35.35	34	35.50	34	35.40	34	35.50
		0.01	34	36.10	34	35.75	34	35.45	33	35.20	34	35.45
		0.1	35	36.05	33	35.80	34	35.80	34	35.45	33	34.95
	0.95	0.00001	34	35.80	33	35.25	34	35.30	33	35.35	34	35.45

		0.0001	34	35.75	34	35.35	34	35.70	34	35.35	34	35.05
		0.001	35	35.95	34	35.40	34	35.25	34	35.30	34	35.45
		0.01	34	35.85	34	35.50	34	35.35	34	35.25	33	35.35
		0.1	35	36.00	34	35.90	35	35.65	34	35.40	34	35.30
0.75	0.05	0.00001	35	35.90	34	35.60	34	35.50	35	35.50	34	35.35
		0.0001	34	35.75	35	35.65	34	35.55	34	35.60	34	35.50
		0.001	34	35.80	34	35.70	34	35.25	33	35.35	33	35.00
		0.01	34	35.60	34	35.30	33	35.45	34	35.50	34	35.15
		0.1	34	35.60	34	35.35	35	35.80	35	35.80	34	35.45
	0.25	0.00001	34	35.85	33	35.50	33	35.35	34	35.20	34	35.20
		0.0001	34	36.10	34	35.40	33	35.20	34	35.40	34	35.45
		0.001	34	36.00	34	35.45	33	35.40	34	35.30	33	35.45
		0.01	34	35.95	34	35.70	35	35.25	34	35.35	33	35.45
		0.1	35	35.95	34	35.35	34	35.55	34	35.15	34	35.25
	0.5	0.00001	34	35.65	32	35.65	34	35.45	34	35.45	33	35.10
		0.0001	35	36.10	34	35.70	33	35.50	34	35.30	34	35.25
		0.001	35	35.95	35	35.85	34	35.45	34	35.25	34	35.50
		0.01	35	35.60	33	35.50	33	35.35	35	35.55	34	35.50
		0.1	34	35.50	34	35.95	33	35.40	35	35.75	35	35.65
	0.75	0.00001	35	35.65	34	35.30	35	35.30	34	35.35	34	35.30
		0.0001	34	35.75	35	35.60	35	35.30	33	35.30	34	35.50
		0.001	35	35.65	34	35.70	33	35.30	34	35.30	34	35.35
		0.01	34	35.95	33	35.30	33	35.40	34	35.70	34	35.25
		0.1	34	35.70	33	35.20	33	35.20	34	35.45	34	35.35
	0.95	0.00001	35	36.05	34	35.70	33	35.30	35	35.80	33	35.40
		0.0001	35	35.90	33	35.60	35	35.60	35	35.45	33	35.25
		0.001	34	35.95	34	35.40	33	35.30	33	35.40	34	35.35

		0.01	35	35.85	34	35.70	34	35.65	34	35.75	34	35.55
		0.1	34	35.70	34	35.50	34	35.40	34	35.35	34	35.55
0.95	0.05	0.00001	34	35.50	34	35.65	34	35.45	34	35.35	34	35.05
		0.0001	34	35.85	35	35.65	34	35.45	33	35.25	34	35.45
		0.001	34	35.65	34	35.50	34	35.60	35	35.65	34	35.50
		0.01	35	35.95	34	35.55	33	35.75	34	35.30	33	35.45
		0.1	34	35.70	35	35.65	34	35.50	34	35.50	33	35.25
	0.25	0.00001	34	35.60	34	35.45	33	35.10	32	35.10	33	35.30
		0.0001	33	35.50	34	35.45	33	35.30	33	35.25	35	35.70
		0.001	34	35.35	34	35.55	33	35.40	33	35.30	33	35.60
		0.01	35	35.75	34	35.80	33	35.25	34	35.45	34	35.50
		0.1	34	35.95	34	35.50	34	35.80	34	35.70	34	35.65
	0.5	0.00001	35	35.65	34	35.35	34	35.40	34	35.20	34	35.40
		0.0001	35	35.70	34	35.45	34	35.45	34	35.15	33	35.30
		0.001	34	35.85	34	35.55	34	35.45	34	35.25	33	35.55
		0.01	35	35.95	34	35.60	34	35.15	34	35.35	33	35.45
		0.1	35	36.25	34	35.80	33	35.45	35	35.45	34	35.50
	0.75	0.00001	33	35.60	34	35.80	34	35.20	34	35.70	33	35.55
		0.0001	33	35.55	34	35.60	33	35.45	34	35.40	33	35.20
		0.001	35	35.85	34	35.35	33	35.35	34	35.55	34	35.65
		0.01	34	35.80	33	35.30	35	35.65	34	35.50	35	35.75
		0.1	35	36.00	35	35.55	33	35.50	34	35.60	34	35.45
	0.95	0.00001	34	36.00	34	35.50	33	35.00	34	35.65	35	35.45
		0.0001	34	35.70	34	35.80	33	35.05	33	35.30	34	35.55
		0.001	35	35.85	34	35.50	34	35.35	33	35.15	34	35.50
		0.01	33	35.70	34	35.50	34	35.70	34	35.70	33	35.20
		0.1	35	36.15	35	35.70	33	35.65	34	35.20	34	35.35

APPENDIX C
THE LIST OF PUBLICATIONS AND AWARDS

SELECTED PUBLICATIONS:

Yazan A. Alsariera, Hammoudeh S. Alamri, Abdullah M. Nasser, Mazlina A. Majid, and Kamal Z. Zamli. 2014, "Comparative performance analysis of bat algorithm and bacterial foraging optimization algorithm using standard benchmark functions." *In the 8th Malaysian Software Engineering Conference (MySEC2014)*, 295-300.

Abdullah M. Nasser, Yazan A. Alsariera, Kamal Z. Zamli, and B AlKazemi. 2014, "Late acceptance hill climbing based strategy for addressing constraints within combinatorial test data generation". *In the 8th SOFTEC Asia 2014 Conference (SOFTEC2014)*, 12-16.

Yazan A. Alsariera and Kamal Z. Zamli. 2015, "A bat-inspired strategy for t-way interaction testing", *Journal of Advanced Science Letters, American Scientific Publishers*, **21**(7), 2281-2288.

Yazan A. Alsariera, Mazlina A. Majid, and Kamal Z. Zamli. 2015, "A bat-inspired strategy for pairwise testing", *Journal of Engineering and Applied Sciences, ARPN*, **10**(18), 8500-8506.

Kamal Z. Zamli, Yazan A. Alsariera, Abdullah B. Nasser, and Abdul Rahman A. Alsewari. 2015, "On adopting parameter free optimization algorithms for combinatorial interaction testing", *ARPN Journal of Engineering and Applied Sciences*, ARPN Pakistan, **10**(19), 8987-8994.

Yazan A. Alsariera, Mazlina A. Majid, and Kamal Z. Zamli. 2015, "SPLBA: An interaction strategy for software product line system configuration using bat-inspired algorithm", *In the 4th International Conference on Software Engineering and Computer Systems (ICSECS2015)*, 148-153.

Yazan A. Alsariera, Mazlina A. Majid, and Kamal Z. Zamli. 2015, "An interaction testing case study using bat-inspired t-way strategy", *In the 8th SOFTEC Asia 2015 Conference, (SOFTEC2015)*, 1-4.

Abdullah B. Nasser, Yazan A. Alsariera, AbdulRahman A. AlSewari, Kamal Z Zamli. 2015. "A cuckoo search based pairwise strategy for combinatorial testing problem. A cuckoo

search based pairwise strategy for combinatorial testing problem", *Journal of Theoretical and Applied Information Technology (JATIT)*, **82**(1), 154-162.

Abdullah B. Nasser, Yazan A. Alsariera, AbdulRahman A. AlSewari, Kamal Z Zamli. 2015, "Assessing optimization based strategies for t-way test suite generation: The case for flower-based strategy", *In the International Conference on Control System, Computing and Engineering (ICCSCE2015)*.

Yazan A. Alsariera, Abdullah M. Nasser, Kamal Z. Zamli, 2016, "Benchmarking of bat-inspired interaction testing strategy". *International Journal of Computer Science and Information Engineering*, **7**(1), 71-79.

Yazan A. Alsariera, Hammoudeh S. Alamri, Alaa A. Al-Omoush and Kamal Z. Zamli. 2017, "A real-world test suite generation using the bat-inspired t-way Strategy", *In the 10th SOFTEC Asia 2017 Conference, (SOFTEC2017)*, 38-43.

AWARDS

Best paper award, 2014. *In the Malaysian Software Engineering Conference* (*MYSEC 2014*).

Gold medal, 2015, In *the Creation, Innovation, Technology and Research Exposition* (*Citrex2015*), *Universiti Malaysia Pahang (UMP).*

Gold medal, 2015, *In the International Conference and Exposition on Inventions by Institutions of Higher Learning* (*PECIPTA15*).