UNIVERSITI SAINS MALAYSIA

Laporan Akhir Projek Penyelidikan Jangka Pendek

Investigating T-Way Test Data Reduction Strategy Using Particle Swarm Optimization Technique

by
Assoc. Prof. Dr. Kamal Zuhairi Zamli
Mr. Bestoun S. Ahmed

2012

203/PELECT/6071186 - dah tutup akaun 1 F0520

BORANG FRGS – P3(R)

Kod Projek :	FRGS/FASA1-2009/(BIDANG)/(NAMA IPT)/(NO.RUJ. KPT)

FINAL REPORT
FUNDAMENTAL RESEARCH GRANT SCHEME (FRGS)
Laporan Akhir Skim Geran Penyelidikan Asas (FRGS) IPT
Pindaan 1/2009

A

RESEARCH TITLE : **Investigating T-Way Test Data Reduction Strategy Using Particle Swarm Optimization Technique**
Tajuk Penyelidikan

PROJECT LEADER : **Assoc Prof Dr Kamal Zuhairi bin Zamli**
Ketua Projek

PROJECT MEMBERS (including GRA) : 1. **Bestoun S. Ahmed**
2.
Ahli Projek

PROJECT ACHIEVEMENT *(Prestasi Projek)*

B

ACHIEVEMENT PERCENTAGE			
Project progress according to milestones achieved up to this period	**0 - 50%**	**51 - 75%**	**76 - 100%**
Percentage			100%

RESEARCH FINDINGS		
Number of articles/ manuscripts/ books	**Indexed Journal**	***Non-Indexed Journal***
	3 ISI 3 SCOPUS	
Paper presentations	**International**	***National***
	3 IEEE Conf. papers	2
Others (Please specify)		

HUMAN CAPITAL DEVELOPMENT			
Human Capital	**Number**		Others (Please specify):
	On-going	**Graduated**	
PhD Student		1 (graduate April 2012)	
Masters Student		3 Mixed Mode	
Undergraduate Students			
Temporary Research Officer	1		
Temporary Research Assistant			
Total	5		

EXPENDITURE *(Perbelanjaan)*

C

Budget Approved *(Peruntukan diluluskan)* : **RM 48,000.00**
Amount Spent *(Jumlah Perbelanjaan)* : **RM 35,024.59**

Balance *(Baki)* : **RM 12,975.41**
Percentage of Amount Spent : 72.96%
(Peratusan Belanja)

ADDITIONAL RESEARCH ACTIVITIES THAT CONTRIBUTE TOWARDS DEVELOPING SOFT AND HARD SKILLS
(Aktiviti Penyelidikan Sampingan yang menyumbang kepada pembangunan kemahiran insaniah)

D

International

Activity	Date (Month, Year)	Organizer
Kamal Z. Zamli, Combinatorial Software Testing: From Pairwise and Beyond, Invited tutorial in the IEEE International Symposium on Industrial Electronics and Applications (ISIEA 2010)	3rd October 2010, Park Royal, Penang	IEEE Industrial Electronics

National

Activity	Date (Month, Year)	Organizer
Kamal Z. Zamli, Keynote Speaker, Software Testing Conference 2011, SOFTEC2011	6th July 2011, Prince Hotel, KL	Malaysian Software Testing Board

PROBLEMS / CONSTRAINTS IF ANY *(Masalah/ Kekangan sekiranya ada)*

E None

RECOMMENDATION (*Cadangan Penambahbaikan*)

F None

ABSTRACT (*Abstrak*)

G Recently, researchers have started to explore the use of Artificial Intelligence (AI)-based algorithms as *t*-way (where *t* indicates the interaction strength) and variable-strength testing strategies. Many AI-based strategies have been developed, such as Ant Colony, Simulated Annealing, Genetic Algorithm, and Tabu Search. Although useful, most existing AI-based strategies adopt complex search processes and require heavy computations. For this reason, existing AI-based strategies have been confined to small interaction strengths (i.e., t≤3) and small test configurations. Recent studies demonstrate the need to go up to *t*=6 in order to capture most faults. This research presents the design and implementation of a new interaction test generation strategy, known as the Particle Swarm-based Test Generator (PSTG), for generating *t*-way and variable-strength test suites. Unlike other existing AI-based strategies, the lightweight computation of the particle swarm search process enables PSTG to support high interaction strengths of up to *t*=6. The performance of PSTG is evaluated using several sets of benchmark experiments. Comparatively, PSTG consistently outperforms its AI counterparts and other existing strategies as far as the size of the test suite is concerned. Furthermore, the case study demonstrates the usefulness of PSTG for detecting faulty interactions of the input components.

Date : 24 July 2012
Tarikh

Project Leader's Signature:
Tandatangan Ketua Projek

COMMENTS, IF ANY/ ENDORSEMENT BY RESEARCH MANAGEMENT CENTER (RMC)
(Komen, sekiranya ada/ Pengesahan oleh Pusat Pengurusan Penyelidikan)

H .....
Good output.
.....

Name:
Nama:

Signature:
Tandatangan:

Date:
Tarikh: 24/7/13

LIST OF OUTPUTS

ISI/SCOPUS Journals

AHMED, B. S., ZAMLI, KAMAL Z. ZAMLI & C. P. LIM 2012. Application of Particle Swarm Optimization for uniform and variable strength covering array construction. Applied Soft Computing Journal, 12(4), pp. [ISI Impact Factor =2.612].1330-1347

BESTOUN S. AHMED, KAMAL Z. ZAMLI, 2011. A variable strength interaction test suites generation strategy using Particle Swarm Optimization. Journal of Systems and Software, 84(12), pp. 2171-2185. [ISI Impact Factor =0.836].

AHMED, B. S., ZAMLI, KAMAL Z. ZAMLI & C. P.LIM 2012. Constructing a t-way interaction test suite using the particle swarm optimization approach. International Journal of Innovative Computing, Information and Control (IJICIC), 8(1A), pp. 431-452. [ISI Impact Factor =1.66].

BESTOUN S. AHMED, KAMAL Z. ZAMLI, 2011. A review of covering arrays and their application to software testing. Journal of Computer Science, 7(9), pp. 1375-1385.(SCOPUS Indexed).

BESTOUN S. AHMED, KAMAL Z. ZAMLI, 2011. The development of a Particle Swarm Based Optimization strategy for pairwise testing. Journal of Artificial Intelligence, 4(2), pp.156-165. (SCOPUS Indexed).

BESTOUN S. AHMED, KAMAL Z. ZAMLI, 2011. A greedy Particle Swarm Based Optimization strategy for t-way testing. Journal of Artificial Intelligence, 5(2), pp.85-90. (SCOPUS Indexed).

Conference Proceedings

BESTOUN S. AHMED, KAMAL Z. ZAMLI, 2009. Motivation for developing a high combinatorial interaction strength testing suite via the Grid Computing approach. Proceedings of theElectrical and Electronic Postgraduate Colloquium EEPC2009, Jawi, Malaysia.

BESTOUN S. AHMED, KAMAL Z. ZAMLI, 2010. PSTG: a t-way strategy adopting particle swarm optimization. Proceedings of the4th Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation, Kota Kinabalu, Borneo, Malaysia. IEEE Computer Society, pp. 1-5.

BESTOUN S. AHMED, KAMAL Z. ZAMLI, 2010. T-way test data generation strategy based on particle swarm optimization. Proceedings of the2nd International Conference on Computer Research and Development, Kuala Lumpur, Malaysia. IEEE Computer Society, pp. 93-97.

BESTOUN S. AHMED, KAMAL Z. ZAMLI, 2011. Comparison of metaheuristic test generation strategies based on interaction elements coverage criterion. Proceedings of theThe 2011 IEEE Symposium on Industrial Electronics and Applications (ISIEA 2011), Langkawi, Malaysia. IEEE Computer Society, pp. 550-554.

AHMED, B. S. & ZAMLI, K. Z. 2011. A greedy Particle Swarm Optimization strategy for t-way software testing. Proceedings of theThe Electrical and Electronic Postgraduate Colloquium (EEPC2011), Dusun Eco Resort, Bentong, Pahang, Malaysia

Human Capital Developments

BESTOUN S. AHMED, Adopting a Particle Swarm-based Test Generator Strategy for Variable Strength and T-Way Testing, PhD (completed March 2012)

JULIANA MD SHARIF, Implementing Seeding and Constraint Mechanism for Pairwise Test Data Generation, MSc Mixed Mode Dissertation (completed June 2010)

NOR HIDAYAH SAAD, Enhancing a Pairwise Test Data Generation to Support High T-Way Interaction, MSc Mixed Mode Dissertation (completed June 2010)

AMIR ABU BAKAR, Enhancing T-Way Strategy with Seedings and Constraints Support, MSc Mixed Mode Dissertation (completed Dec 2010)

Purchase Requisition ▸ Purchase Order ▸ Suppliers ▸ Maintenance ▸ Financials ▸ Coda Info ▸ Reports ▸ Admin ▸

UserCode: ZAIDA / USMKCTLIVE / PELECT **Program Code: Votebook9100** **Current Program : Votebook (Header)**

Current Date : 24/07/2012 12:01:41 PM **Version: 15.03, Last Updated at 15/03/2012** **DB: 13.00, 09/18/2010 VB: 13.01, 03/14/2011** **Switch Language : English /Malay**

Wildcard : eg. Like 100%, Like 10%1, Like %1

Element 1: 203 Element 2: % Element 4: PELECT

Element 5: 6071186 Year: 2012 [redacted]

Detail	Excel	Budget Rule	Budget Control	Account Description	Budget Account Code	Roll over	Budget	Cash Received	Advanced	Commit	Actual	Available	Percentage
Detail	Excel	116	T	Penyelidikan Fundamentals (FGRS)	203.111.0.PELECT.6071186	17,902.14	0.00	0.00	0.00	0.00	0.00	17,902.14	0.00%
		116	T	SubTotal		17,902.14	0.00	0.00	0.00	0.00	0.00	17,902.14	0.00%
Detail	Excel	117	T	Penyelidikan Fundamentals (FGRS)	203.221.0.PELECT.6071186	2,841.30	0.00	0.00	0.00	0.00	0.00	2,841.30	0.00%
Detail	Excel	117	T	Penyelidikan Fundamentals (FGRS)	203.223.0.PELECT.6071186	1,000.00	0.00	0.00	0.00	0.00	0.00	1,000.00	0.00%
Detail	Excel	117	T	Penyelidikan Fundamentals (FGRS)	203.227.0.PELECT.6071186	880.00	0.00	0.00	0.00	0.00	1,780.00	-900.00	0.00%
Detail	Excel	117	T	Penyelidikan Fundamentals (FGRS)	203.228.0.PELECT.6071186	1,500.00	0.00	0.00	0.00	0.00	0.00	1,500.00	0.00%
Detail	Excel	117	T	Penyelidikan Fundamentals (FGRS)	203.229.0.PELECT.6071186	-6,186.20	0.00	0.00	0.00	2,694.92	486.91	-9,368.03	0.00%
		117	T	SubTotal		35.10	0.00	0.00	0.00	2,694.92	2,266.91	-4,926.73	0.00%
		9999		GrandTotal		17,937.24	0.00	0.00	0.00	2,694.92	2,266.91	12,975.41	0.00%

INVESTIGATING T-WAY TEST DATA REDUCTION STRATEGY USING PARTICLE SWARM OPTIMIZATION TECHNIQUE

BY

ASSOC PROF DR KAMAL Z. ZAMLI

FRGS – Final Grant Report

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

USM

2012

Table of Contents

List of Tables

List of Figures and Illustrations

Abstract

Recently, researchers have started to explore the use of Artificial Intelligence (AI)-based algorithms as *t*-way (where *t* indicates the interaction strength) and variable-strength testing strategies. Many AI-based strategies have been developed, such as Ant Colony, Simulated Annealing, Genetic Algorithm, and Tabu Search. Although useful, most existing AI-based strategies adopt complex search processes and require heavy computations. For this reason, existing AI-based strategies have been confined to small interaction strengths (i.e., $t \leq 3$) and small test configurations. Recent studies demonstrate the need to go up to $t=6$ in order to capture most faults. This research presents the design and implementation of a new interaction test generation strategy, known as the Particle Swarm-based Test Generator (PSTG), for generating *t*-way and variable-strength test suites. Unlike other existing AI-based strategies, the lightweight computation of the particle swarm search process enables PSTG to support high interaction strengths of up to $t=6$. The performance of PSTG is evaluated using several sets of benchmark experiments. Comparatively, PSTG consistently outperforms its AI counterparts and other existing strategies as far as the size of the test suite is concerned. Furthermore, the case study demonstrates the usefulness of PSTG for detecting faulty interactions of the input components.

CHAPTER 1

INTRODUCTION

Software testing is an activity that aims to evaluate the capability of a program as well as to determine whether it meets its required results or not [1]. Owing to its usefulness in the software development life cycle, software performance testing comprehends a variety of activities, including stress, isolation, and configuration testing [2, 3]. In each activity, test cases are used in an established test plan to run experiments occupying the software system components. In large software systems, this process is limited by cost because the addition of each test case leads to additional expenditures. This, in turn, leads to the inability of exhaustive testing in performing such a testing process.

Design of Experiment (DOE) has been used to aid the software performance testing [4]. Here, each component of the system is called a "factor," and each test case is called an "experimental run." An experimental run represents a test case to comprehend the system components, where each component is represented by its valid numeric value or configuration [5]. When the system is tested exhaustively, the "full factorial" design of experiment is used [5]. However, when the system is large and the full factorial design is not desirable, the "fractional factorial" design is used to reduce the experimental run to a subset of the full factorial design. The fractional factorial design is used with systems of numeric factors; conversely, systems with categorical factors cannot use this method for experiments [6].

The D-Optimality design, on the other hand, has been used with systems, including categorical factors, to reduce the experimental run by selecting a subset of runs from the full factorial [7, 8]. Instead of a purely random selection of subsets of experimental runs from the full factorial design, the use of the D-Optimality design method in experiments leads to the production of experimental runs that are closer to full factorial design [4].

Recently, an alternative design based on Covering Array (CA) has been used for the approximation of full factorial design [4]. Compared with D-Optimality, empirical evidence demonstrates that CA produces better results than full factorial approximation experiments [4, 8]. In such a design, each *t*-set of factors (or system components) is covered by a set of experimental runs (at least once) to form a CA.

The use of CAs has proven to be adequate and effective in several applications, including drug screening, regulation of gene expression, data compression, code coverage and GUI testing [9-14]. Motivated by the effectiveness of CAs, a number of recent studies have focused on the construction of CAs for combinatorial interaction testing using *t*-way strategies, where *t* signifies the interaction strength of the component. These strategies aim to optimally reduce the number of test cases (i.e., the number of rows in the CA) by ensuring that each test case greedily covers the required *t*-interactions (or *t*-set of factors) at least once for a typically large space of possible test values. This mechanism uniformly covers *t*-interactions of the system components to generate test cases. However, often, the interactions between parameter components are typically non-uniform [15, 16]. As an example, a system with an overall component

values of two-way (pairwise) strength might have a subset of higher strength than the component values for the test [17]. Therefore, the strength might vary and be non-uniform during the testing process of the system component values. Taking both cases (i.e., uniform and variable interaction strength) as an NP hard computational optimization problem [16, 18, 19], many strategies based on Artificial Intelligence (AI) have been developed. Recent researches demonstrate that strategies based on Genetic Algorithm (GA), Ant Colony Algorithm (ACA), Simulated Annealing (SA), and Tabu Search (TS) can effectively generate small-sized CAs.

1.1 Problem Statements

Although useful, most existing AI-based t-way testing strategies require complex computations (i.e., in terms of the need to deal with mutations, crossovers, and the local minima problem [17, 20-22]). For these reasons, existing AI-based t-way testing strategies have been confined to small interaction strengths (i.e., $t<3$) and small test configurations [15, 23-25]. To be effective, recent studies and empirical evidence demonstrate the need to go up to $t=6$ in order to capture most faults in a software module [10, 26-28].

Particle Swarm Optimization (PSO) is known for the simplicity of its algorithm structure over other optimization methods [29-32]. PSO is also requires lightweight computations. In this research, we investigate the competitiveness of our proposed Particle Swarm-based t-way Test Generator (PSTG) based on PSO for uniform and variable strength CA generation. Unlike other existing AI-based t-way testing strategies,

the use of PSO leads to lightweight computation in PSTG, thus, enabling it to support high interaction strengths of up to t=6 [33, 34].

1.2 Report Aim and Objectives

The aim of the research is to design, implement, and evaluate a new interaction testing strategy, called Particle Swarm Test Generator (PSTG), for constructing t-way and variable interaction strength test suites based on Particle Swarm Optimization. To realize this aim, the following objectives are adopted:

i. To investigate the application of Particle Swarm Optimization for PSTG's design and implementation in order to support t-way and variable-strength test suites construction.
ii. To investigate and evaluate the performance of the PSTG strategy against other computational and AI-based strategies in term of the generated test suite size.
iii. To investigate and evaluate the effectiveness of the test suites generated by the PSTG strategy for interaction fault detection.

1.3 Methodology of the Research

Overall, the research's methodology is divided mainly into three phases.

i. Literature review: in this phase, the literature survey is undertaken to establish the state-of-the-art on interaction testing. The literature starts by reviewing the importance of the software testing in the software quality assurance process. By establishing this importance, the existing sampling and test design techniques are reviewed also and the importance of the interaction testing as complementary technique in software test design is established. Then, the existing literature of interaction testing strategies is reviewed to identify the features and drawbacks of the strategies and techniques. Based on the literature review survey, the requirement of the research is established in this phase. From the requirement, how the PSO, *t*-way, and variable-strength algorithms will be implemented is decided here.

ii. Design and Implementation: here, the adoption of PSO is established and the required algorithms are decided. Then, the complete algorithms making up the PSTG strategy are designed, implemented, and optimized in this phase. In addition, the parameter tuning of the strategy is performed here also.

iii. Evaluation, Benchmarking, and Case study: experiments with well-known benchmarking configuration as well as a case study are undertaken in this phase to investigate and evaluate the performance and effectiveness of the strategy.

To illustrate how the aforementioned phases are related, Figure 1-1 summarizes the research's activities.

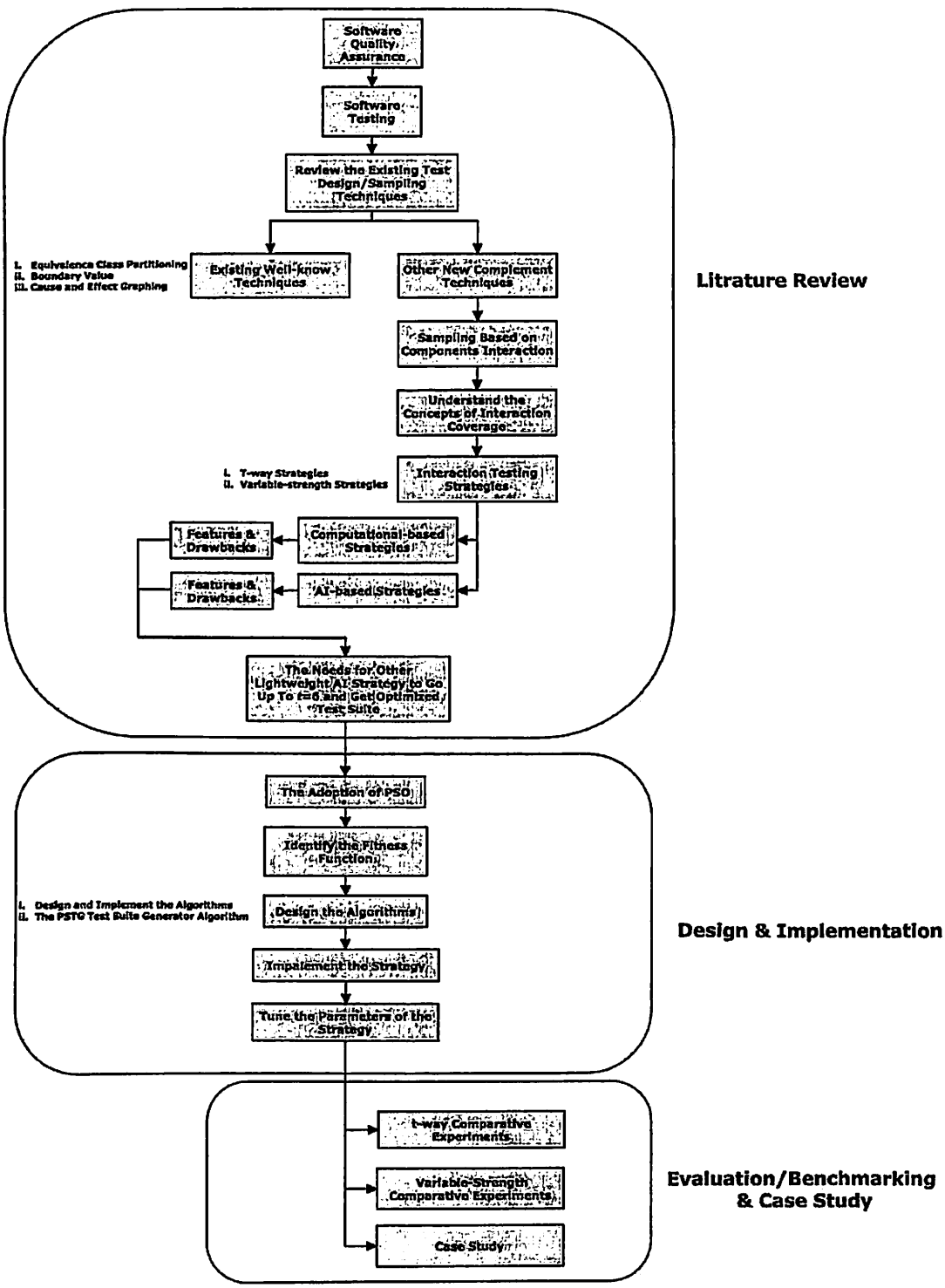


Figure 1-1 The Research's Activities and Flow

1.4 Report Outline

This rest of the report is organised into four other chapters as follows.

Chapter 2 presents theoretical framework along with a survey of existing t-way strategies. Towards the end of Chapter 2, an analysis of existing work is presented which provides the requirements and justification for the development of PSTG.

Chapter 3 discusses and justifies the detailed algorithms and implementation for PSTG based on the requirements from Chapter 2. Additionally, this chapter also elaborates on tuning of the PSTG parameters.

In Chapter 4, a detailed account for evaluating PSTG is presented. Here, PSTG will be compared against existing strategies in terms of the number of generated test data.

The conclusion of this work is given in Chapter 5, where the achievements, contributions and problems are summarised. Conclusions are drawn from the experience gained from this work and the significance of findings along with a consideration for future work.

CHAPTER 2

LITERATURE REVIEW

The previous chapter has established the needs for a new (uniform and variable strength) strategy that is sufficiently lightweight in order to cater for high interaction strength up to 6. In doing so, the previous chapter has also advocated the use of AI-based strategies based on Particle Swarm Optimization (PSO) called PSTG.

In this chapter, the development of PSTG will be further justified by giving an overview of the theoretical framework and a survey of existing literature. This survey and analysis is then used to provide the requirements for PSTG. Finally, this chapter closes by providing a short summary.

2.1 Theoretical Framework

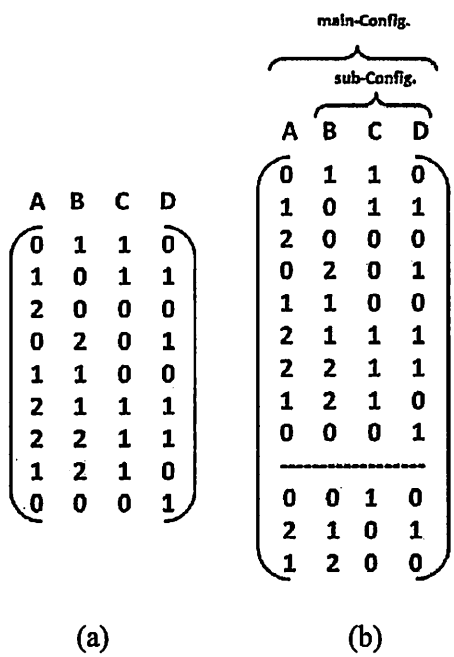
In order to illustrate the basic of t-way testing, there is a need to understand the theoretical framework on Covering Arrays (CA). Originally, the CA has emerged to complement Orthogonal Array (OA) limitations. An OA_λ (N; t, k, v) of strength t is an array of size N and k components with v values, in which for every $N \times t$ sub-array, the t-interaction elements occur exactly λ times, and $\lambda = N/v^t$ [35, 36]. It has been shown that the OA is often too restrictive because it requires the component values to be uniform [9]. To complement the OA construction and to overcome its limitation, the CA has been introduced. The notation CA_λ (N;t,k,v) represents an array of size N with v values, such that every $N \times t$ sub-array contains all ordered subsets from the v values of size t at least λ times [16, 37], and k is the number of

components. To cover all *t*-interactions of the components, it is normally sufficient to occur once in the CA. Therefore, with λ=1, the notation becomes CA (N;t,k,v). When the CA contains a minimum number of rows (*N*), it can be considered an optimal CA according to the definition in eqn. (1) [9].

$$CA(t,k,v) = min\{N: \exists\, CA_{\lambda}(N;t,k,v)\} \qquad (1)$$

However, when the number of component values varies, this can be handled by the Mixed Covering Array (MCA) (N;t,k,(v_1,v_2,...v_k)) [38]. The notation can also be represented by MCA (N;t,k,v^k).

With the availability of these notations, *t*-way test suites can effectively be abstracted. For example, Figure 2-2 (a), represents an MCA (9; 2, 3^2 2^2) of size 9 (i.e., nine test cases) for a system with four components (two components having three values and two components having two values) to cover two-way interactions.



(a) (b)

Figure 2-1 The Representation of Uniform and Variable Strength CAs

With the emergence of different applications of CA and MCA, the Variable Strength Covering Array (VSCA) notation has been introduced as another variant of CA and MCA [16, 39]. A VSCA (N; t,k, (v_1, v_2, ..., v_k) , C) is an $N \times k$ CA or MCA of strength t containing C, a vector of the CA or MCA, and a subset of the k columns each with a strength > t [38]. As an example, to cover the three-way interactions among B, C, and D, we need to add 12 more tests to the two-way MCA in Figure 2-2 (a). However, when using these test cases generated for the two-way MCA, we need to add only three more tests for a total of twelve. The last three rows in Figure 2-2 (b) depict these test cases. Therefore, the resulting VSCA continues to cover the two-way interactions among A,

B, C, and D, i.e., under the "main-config" bracket. In addition, the VSCA covers the three-way interactions among B, C, and D, i.e., under the "sub-config" bracket.

Having described the notations, the following section presents a survey of existing studies on constructing uniform and variable strength CAs in order to reflect the current progress and achievements thus far in the literature.

2.2 Literature Review

Many strategies have been developed to construct uniform and variable strength CAs. The construction methods of the uniform strength CAs came first. Based on those construction methods, the ideas of constructing the variable strength CA have emerged. The next sub-sections review these two construction methods.

2.2.1 Uniform-Strength Covering Array Construction

There are two main methods for the construction of CAs, namely: algebraic [9, 39] and computational methods [39, 40]. Algebraic methods are based on the construction of the OA. The OA is derived from the extensions of mathematical functions. Despite its usefulness, the OA is too restrictive because it exploits mathematical properties, thereby requiring the components and values to be uniform. To overcome this limitation, the Mutual Orthogonal Array (MOA) [41] has been introduced to support non-uniform

values. However, a major drawback exists for both MOA and OA, i.e., a feasible solution is only available for certain configurations [42].

In most cases, computational methods generate all possible interactions. Test cases are then generated to cover these interactions. The method of generating candidate test cases is the main differentiation among the generation strategies. There are two main ways for constructing test cases computationally: one-test-at-a-time or one-parameter-at-a-time [39, 40, 42]. In the former case, the strategy candidate a single or a set of complete test cases per iteration then searches for the test case that covers most of the generated interactions for the CA. Based on this framework, a number of strategies have been developed in previous studies. The most well-known strategies in this approach are Automatic Efficient Test Generator (AETG) [43], mAETG [38], Pairwise Independent Combinatorial Testing (PICT) [44], Deterministic Density Algorithm (DDA) [45, 46], Classification-Tree Editor eXtended Logics (CTE-XL) [47, 48], Test Vector Generator (TVG) [49, 50], Jenny [51], Test Configuration (TConfig) [52], and Intelligent Test Case Handler (ITCH) [53].

In the case of one-parameter-at-a-time, the CA is constructed incrementally by horizontal extension. Each time a component is added to the CA, the strategy performs a coverage check and chooses the best value of the component. If the horizontal extension is performed and some interaction elements have not been covered, the vertical extension immediately begins to cover the uncovered interactions. The CA is constructed completely when these two algorithms are performed. In-Parameter-Order-

General (IPOG) [54] and its improvements, i.e., IPO-s [55], and IPOG-D [42] are three of the most recent strategies that have adopted this approach for the construction of uniform-strength CAs.

As a main part of the computational one-test-at-a-time construction method, significant efforts have emerged to adopt AI-based strategies for CA generation. Thus far, SA, TS, GA, and ACA have been successfully implemented for small-scale interaction strengths [25, 39, 56].

Stardom [56] first implemented SA, GA, and TS to support pairwise (two-way) interactions. The experimental results showed that, because of its algorithmic complexity in finding good solutions, the GA is the least effective as compared with SA and TS. In addition, TS is effective in constructing test cases where the search space is small, whereas SA performs better with larger search spaces. This implementation of TS has also been used by K. Nurmela [57] for pairwise testing and has produced similar results.

Cohen [38] later developed and implemented SA to support up to three-way interactions. The results showed that, in comparison with the greedy search technique of TCG and AETG, SA performed better in generating smaller-sized CAs for pairwise interactions. However, in the case of strength three, SA did not perform as well as compared with the existing algebraic approaches [17]. Hence, the results indicated that

SA is more effective than other approaches for finding optimal sizes in cases of small strengths.

Toshiaki *et al.* [25] also developed and implemented the GA to support up to three-way interactions. In addition, Toshiaki *et al.* employed ACA to support up to three-way interactions. In both cases, a "compaction algorithm" that merges the CA rows for optimality, and further optimizes the resulting CA of the algorithm. Their results have been compared with those from AETG, IPO, and SA. Although GA and ACA outperform AETG, SA outperforms GA and ACA in all cases for strengths two and three. Meanwhile, as demonstrated by Afzal *et al.* [17], it is noticeable that the results of the GA did not match with those produced by Stardom [56]. This indicates that the GA performs poorly for CA generation, although several attempts have been made to modify the algorithm structure.

Although the existing AI-based strategies appear to perform well, a closer look reveals some limitations in terms of complexity of both the search process and algorithm structure. As an example, the large random search space and the update rule of SA make the search process computationally intensive, leading to an increase in computation time, especially when the interaction strength grows up (i.e., t>3) [17, 38, 39, 58]. This problem can be seen clearly in the complexity of the GA crossover and mutation processes, and the increasing numbers of ants in ACA when the problem grows. Similarly, TS suffers from the same problem when it keeps and updates its Tabu

list sets. For these reasons, most strategies implementing SA, GA, and ACA have been limited to small configurations with strengths two and three [17, 25].

2.2.2 Variable Strength Covering Array Construction

A number of existing strategies have started to support VSCA construction (e.g., PICT, TVG, and ITCH). In addition to these computational strategies, a number of AI-based strategies have emerged. Wang *et al.* [24] adopted the DDA algorithm for a variable strength strategy called Density. Wang *et al.* [24] also adopted the IPO algorithm for a variable strength strategy called ParaOrder. This, in turn, has motivated the IPOG research group to add the support of variable strength in the ACTS tool implementation [59, 60].

Cohen *et al.* [15] also developed SA to support VSCA construction. The results reported in their study focused on strengths of two and three only. Moreover, the only published results for this strategy in the case of variable strength are three test configurations.

Xiang *et al.* [23] adopted an improved version of ACA in a strategy called Ant Colony System (ACS) to support VSCA construction. In their study, a "compaction algorithm" was also used to merge the rows in order to further optimize the final VSCA. The results were focused on strengths of two and three only, as indicated in Cohen [15] and

Wang [24]. Overall, SA generated better sizes, whereas ACS achieved comparative results in some cases.

As in the case of uniform-strength, the existing AI-based strategies appear to perform better than other strategies for VSCA construction. However, these strategies often adopt exhaustive search processes with complex algorithm structures. This explains the limited support of up to strengths two and three only.

2.3 The Adoption of PSO for t-way Testing

Enhancing and complementing existing work on AI approaches as *t*-way testing strategies, this research deploys PSO to derive a strategy for uniform and variable strength CA construction. PSO has been demonstrated as an efficient optimization method for many problems [20, 61, 62]. In most cases, PSO does not suffer from the difficulties encountered by other AI techniques, e.g., [21, 30, 31, 61]. Compared with other AI-based techniques, PSO differs in three main points: recombination, mutation, and selection, as follows.

With respect to recombination, PSO does not have a direct recombination operator; despite the fact that stochastic acceleration of a particle towards its previous best position resembles the recombination procedure of other techniques. Instead of recombination, PSO manages information exchange only between the particle's possession experience and the experience of the best particle in the swarm. On mutation, PSO has the advantage of not using evolutionary operators such as crossover

and mutation [32], thereby enabling a lighter computational load. Concerning selection, PSO does not use the survival of the fitness concept, as it does not use direct selection. Therefore, during optimization, particles with lower fitness values can survive and are able to visit any point of the search space [63].

In addition to the above features, PSO is computationally inexpensive because its requirements for memory and CPU speed are low. PSO does not need the calculation of derivatives from other particles, and has few parameters that need to be tuned [64]. Moreover, PSO does not require information of the objective function under testing; it requires only the value, which is used within primitive mathematical operators, hence, leading to a low computation time [63].

The origin of PSO dates back to 1995, when it was first developed by Eberhart and Kennedy as an optimization technique [65] inspired by the swarm behavior of fish and bird schooling in nature. Initially, the main idea was to simulate the unpredictable choreography of a flock of birds. Based on observation of evolutional aspects of the PSO algorithm, it has been realized as an optimizer. Indeed, PSO has received much attention as an optimizer that is applicable to many fields of engineering [63].

A random population of solutions, in which each likely solution is assigned a randomized velocity, initializes the global version of PSO. The likely solutions, called particles, are then "flown" through the problem space iteratively. Each particle remembers its coordinates in the solution space where it has its best solution thus far,

which is called *pBest*. In addition to the best value, the PSO algorithm tracks the overall best value and its location obtained by any particle in the population, which is called *gBest*. When each particle keeps track of the local best solution *lBest* and the neighborhood, in addition to *pBest*, the process is known as the local version of PSO. Figure 2-3 summarizes the PSO algorithm.

```
Begin
Initialize an array of particle's population with random positions and velocities
in D-dimension;
Evaluate fitness for each particle;
While termination criteria is not met do {
   For each particle {
      Modify the velocity according to Eq (2);
      Modify the position according to Eq (3);
      Update the particle's personal best using the update rule;
      }
   For each particle {
      Update the lBest;
      Update the gBest;
      }
   }
End;
```

Figure 2-2 The PSO Algorithm

Taking a D-dimensional search space, the velocity and position of the i^{th} particle in the d^{th} dimension are updated according to the following rules [32]:

$$V_{j,d}(t) = w\, V_{j,d}(t\text{-}1) + c_1\, r_{j,d}(pBest_{j,d}(t\text{-}1) - X_{j,d}(t\text{-}1)) + c_2\, r'_{j,d}(lBest_{j,d}(t\text{-}1) - X_{j,d}(t\text{-}1)) \quad (2)$$

$$X_{j,d} = X_{j,d}(t\text{-}1) + V_{j,d}(t) \quad (3)$$

where t is the iteration number or time, d is the dimension of the j particle index, (c_1, c_2) are the acceleration coefficients that adjust the weight between components, w is the inertia weight, and (r, r') are two random factors, which are two random real numbers in the range of (0,1). According to the updated rule, each particle updates its velocity for better movement around the search space, and the new velocity is used to find a new position for the particles, depending on a cost factor that controls this movement.

In addition to the standard version of PSO, the discrete version of PSO (DPSO) came into existence because of different applications [61, 66]. Using DPSO, some changes must be made to facilitate the adaptation of discrete space [67]. Owing to the restriction of discrete component values in the system under test, we have selected DPSO in this work. In fact, the DPSO procedure is similar to standard PSO. However, in standard PSO, an array of particles is initialized, in which the values of the particles are

continuous and are not restricted. In DPSO, however, the values of the particles are restricted [66].

Based on the aforementioned features and considering the generation of t-way and variable strength test suites as optimization problem, this work has considered PSO as the basis of PSTG.

2.4 Summary

In this chapter, the theoretical framework and the survey of t-way strategies for both uniform and variable interaction testing have been discussed. The adoption of PSO has been justified including its features and advantages to be implemented for an interaction testing strategy.

Building on that, the next section addresses the adoption of PSO in the proposed strategy, PSTG. The next chapter illustrates the algorithms implemented in the strategy and how PSO is implemented for PSTG.

CHAPTER 3

DEVELOPMENT OF THE PSTG STRATEGY

In the previous chapter, the theoretical background, notations, and definitions for interaction testing have been presented. An extensive review of the existing literature on *t*-way and variable-strength strategies and their applications in real world has also been given. Moreover, justifications for the adoption of PSO in the proposed PSTG strategy are provided. This chapter discusses the design and implementation of the PSTG strategy including its corresponding algorithms. The chapter also describes how to apply PSO in the strategy and how to choose and set its design parameters properly.

3.1 Overview of PSTG

This section describes the application of PSO in our proposed PSTG strategy. PSO is adopted in an algorithm named the CA Generation Algorithm. The algorithm generates test cases based on PSO in a greedy fashion. The fitness function is used to choose the best particle. In our strategy, the fitness function is the number of interactions that the particle can cover. Hence, in order to compute the fitness function for each particle, we must first generate the Interaction Elements (IEs) and provide a mechanism for computing the fitness function undertaken by the IE Generation Algorithm. Therefore, the next sub-section describes the IE Generation Algorithm, after which the CA Generation Algorithm is described.

3.1.1 The IE Generation Algorithm

As previously described, the number of covered IEs determines the fitness function in PSTG. As such, generation of the IEs is required. To illustrate the generation algorithm clearly, the left hand side of Figure 3 shows the flowchart of the IE generation algorithm, whereas the right hand side of Figure 3 shows an example of the generation process described in a step-by-step manner.

The algorithm receives the input of PSTG in the form of the CA, MCA, or VSCA. In all cases, the algorithm scans the input to determine whether the input is of a uniform or variable strength. In the case of a variable strength, the algorithm identifies and separates the main and sub-configurations. However, in a uniform-strength case, the algorithm skips this step directly and proceeds to the next step. For a better understanding, we take a VSCA configuration as the example in Figure 3-1.

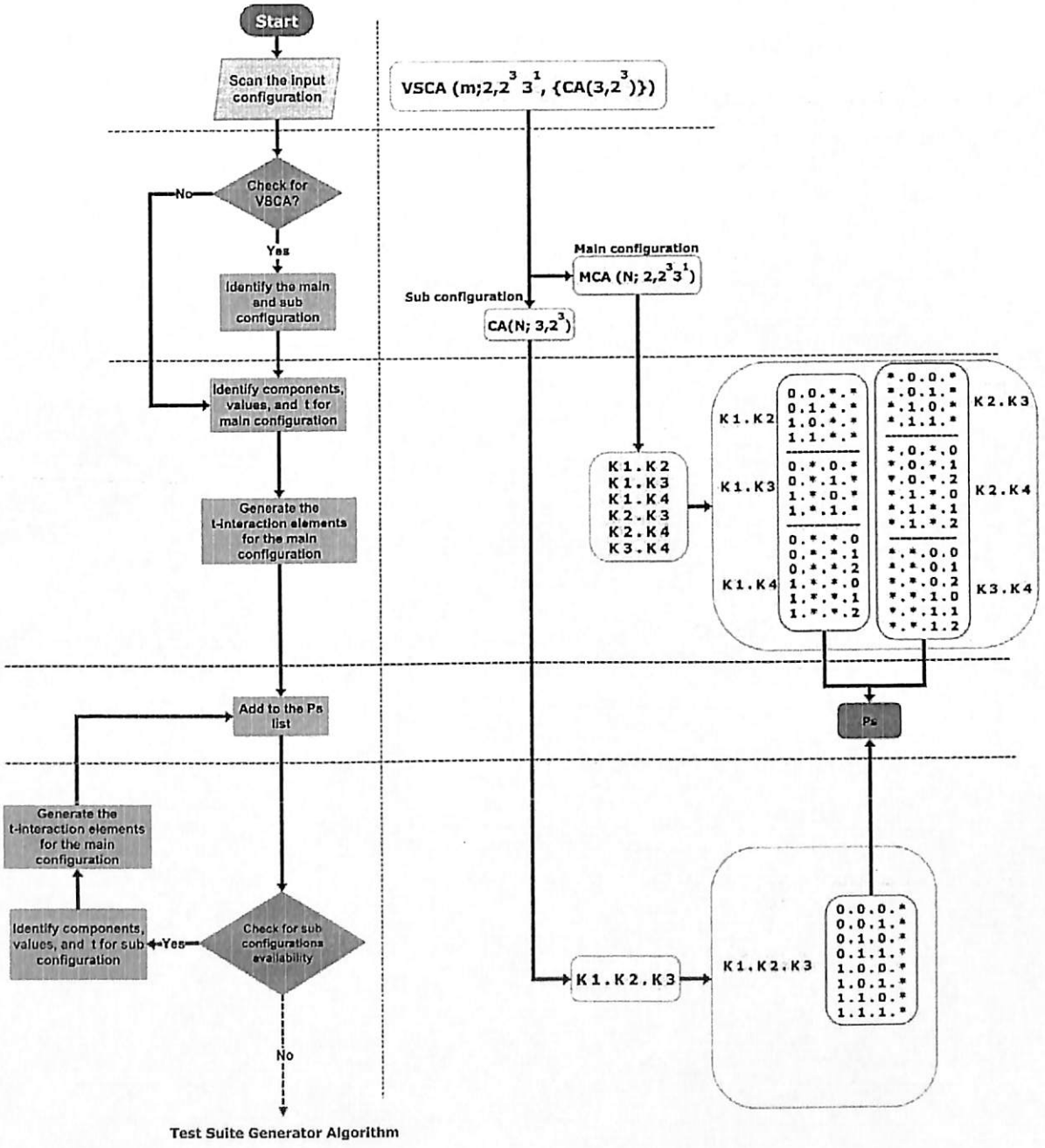


Figure 3-1 IE Generation Algorithm of PSTG

After identifying the main and sub-configurations, the algorithm further identifies the strength, components, and the values of each component for each configuration. From the information gathered, the algorithm first generates the component combinations. Based on these combinations, the IEs are generated. In the example in Figure 3, the variable strength configuration VSCA (m; 2, $2^3 3^1$, {CA (3, 2^3)) has four components with strength two for the main configuration and three components with strength three for the sub-configuration. The first three components have two values (0 and 1), and the last component has three values (0, 1, and 2). The first three components are used for the sub-configuration with strength three.

For each configuration, all binary number possibilities are generated. The number of digits for the binary number is equal to the number of components, i.e., four digits for the main-configuration and three digits for the sub-configuration here. Then, based on the strength, the binary numbers are selected. The strength of the main-configuration is two; thus, the binary numbers that contain two ones are selected. As an example, binary number 0101 refers to the k2. k4 combination.

Based on the generated combinations of the components, the IEs are generated accordingly. In the above example, strength two of the components has six possible combinations. For combination 0101, whereby the second and fourth components are available (i.e., k2 and k4), there are 2×3 possible IEs between them. For each combination, the value of the corresponding component is included in the IEs. When the component is not available (i.e., in case of 0 in the combination), the corresponding

values are marked as "don't care". This process is iteratively repeated for the other five combinations for the main-configuration, i.e., (k1, k2), (k1, k3), (k1, k4), (k2, k3), and (k3, k4) as well as for the sub-configuration, i.e., (k1, k2,k3).

Upon the generation of the IEs, each group of elements is stored in an indexed list called *Ps* (Figure 3). The indexing reference of each combination is also stored in a deferent list. The indexing reference represents two integer numbers that refer to the start and the end of a given combination of IEs in the *Ps* list. Thus, the search for a given IE can be performed efficiently and quickly.

3.2 The CA Generation Algorithm

The CA generation algorithm is performed immediately after the generation of the *Ps* list. The use of the *Ps* list is essential for computing the fitness factor. The fitness factor is used with PSO in a greedy fashion to identify the better particles. Owing to the discrete component values, DPSO is adopted.

The CA generation algorithm is initialized by generating a random swarm search space. The swarm search space takes the form of a D-dimensional vector, $X_j = (X_{j,1}, X_{j,2}, X_{j,d} \ldots\ldots, X_{j,D})$, where each dimension represents a component and contains integer numbers between 0 and (v_i) (i.e., the number of values of the i^{th} component). The

velocity of each particle is also simultaneously initialized with random integer numbers (zero in our case).

As previously discussed, during the iteration of the algorithm, the velocity and the position of the particles are updated according to equations (2) and (3) (i.e., when the particle moves around in the search space). During the update process, there is a possibility of producing non-integer velocities for the particles, leading to the production of non-integer positions. To avoid such a situation, the velocity is rounded to the nearest integer number.

Owing to the continuous space of the velocity, there is a possibility of producing velocity values that cause the particles to fly outside the swarm during the iteration and update process. As such, boundary conditions to restrict the values of the velocity to both lower and higher bounds must be established. The boundary condition is set to $V_{i\ max} = v_i/2$ because the particle dimensions are between 0 and vi. This is in accordance with the recommendations in [68] and [61]. Hence, the velocity bound is between $[-V_{i\ max}, V_{i\ max}]$.

By restricting the velocities to a certain boundary, the particles are also required to avoid the appearance of invalid values of the components. Based on the previous studies, there are three different boundary conditions for DPSO: invisible, reflecting, and absorbing walls [32, 68]. In invisible walls, when the particle goes outside the boundary and an invalid value of the particle appears, the corresponding fitness value is

not computed. In reflecting walls, the motion of the particle is reversed when a particle reaches the boundary. As a result, the particle is reflected back to the search space. In absorbing walls, the velocity changes to zero in the dimension where the particle goes outside.

In our case, the above-mentioned boundary conditions require time for generation, and lead the particle away from the component values. As such, we have configured our boundary conditions in such a way that when the velocity reaches a certain dimension bound, it continues its motion with the same velocity, starting from the other bound of that dimension, by resetting the position to the other endpoint. As an example, in case of a parameter with a range of values from 0 to 3, when the position is greater than 3, the position is reset to 0.

Selection of the personal best from the neighborhood represents another important issue. Different topologies have been proposed to find *pBest* from its neighbor. In this research, we have adopted the simplest topology as proposed by [68], i.e., the particles in a swarm matrix or array choose the neighbors next to them.

The CA generation algorithm uses the aforementioned design to generate the final optimized CA. Figure 3-2 summarizes the algorithm. The algorithm receives the *Ps* list from the IE generation algorithm. Then, the algorithm randomly initializes each particle with its associated component values. For each particle in the swarm search space, the algorithm computes the weight of coverage using the check weight function. The check

weight function converts the given test case to its base IE, and returns the number of IEs covered, i.e., when the weight equals six, six IEs can be covered by the candidate test case. When the weight is equal to the maximum weight, i.e., the test case covers all the component interactions of specific values, the algorithm directly considers that test case as a row in the final CA, and removes the covered IEs. Otherwise, based on the result of the check weight function, the algorithm chooses the test case that covers the most IEs to be *lBest.* For the next iteration, the algorithm updates the positions of the particles according to the update rule, considering the *lBest* value that has been achieved so far.

After updating the positions of the particles, the algorithm re-evaluates the particles, and searches for a better *lBest* value. If a better *lBest* value is found, the new value is set as the new *lBest* value replacing the old *lBest* value. This iteration continues until the end, or until a better *lBest* value can no longer be obtained. In this case, *lBest* is set as *gBest*, and the algorithm adds *gBest* to *Ts* of the output test cases, i.e., to form the final CA. At the same time, the covered *t*-interaction elements are removed from *Ps*. The algorithm continues until *Ps* becomes empty. Here, indexing is necessary in order to expedite the process of finding the covered interaction.

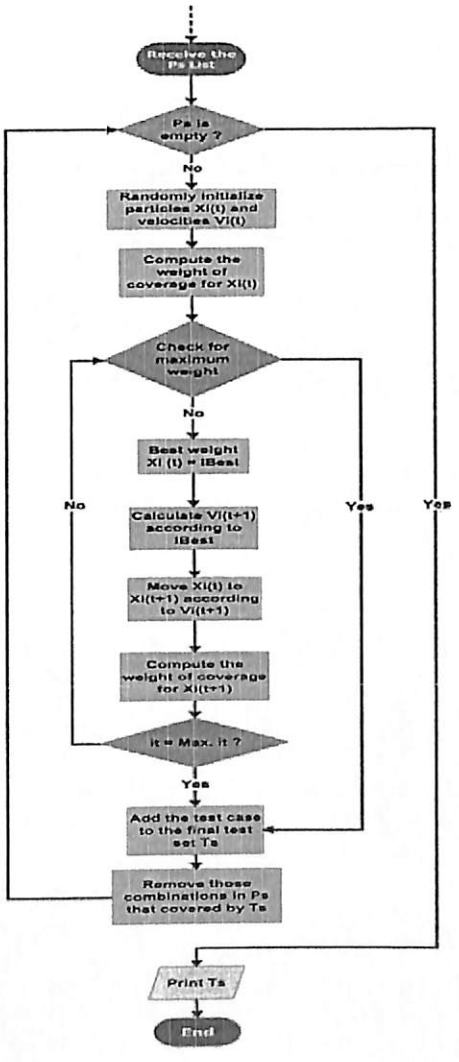


Figure 3-2 A Flowchart of the PSTG CA Generator Algorithm

3.3 PSTG Parameter Setting

Referring to PSO equations in Section 2.3, choosing suitable parameter values for the parameters is necessary for achieving the best performance of PSO in different types of applications, depending on the problem at hand. As discussed in previous studies, selection of these parameters varies from one application to another, depending on the problem [32, 63]. Therefore, in our case, selection of these parameters has to be addressed.

Choosing a large velocity for the particles facilitates global exploration, whereas choosing a small value facilitates local exploration. The inertia weight, w, is regarded as a balance provider between the local and global exploration abilities. A good choice of w results in a reduction of the number of iterations required to locate the optimum solutions. In fact, the inertia weight has been designed to have a better control of the particle velocity.

The learning factors, c_1 and c_2, control the stochastic acceleration process of the particles and attempt to pull each particle towards the best achieved solutions [32, 63]. Choosing a low value for these parameters leads the particle to move away significantly from the target region. On the other hand, choosing a high value for these parameters

optimal sizes of 19, and 29, respectively [71, 72]. Figures 3-3 and 3-4 show the results of the experiments.

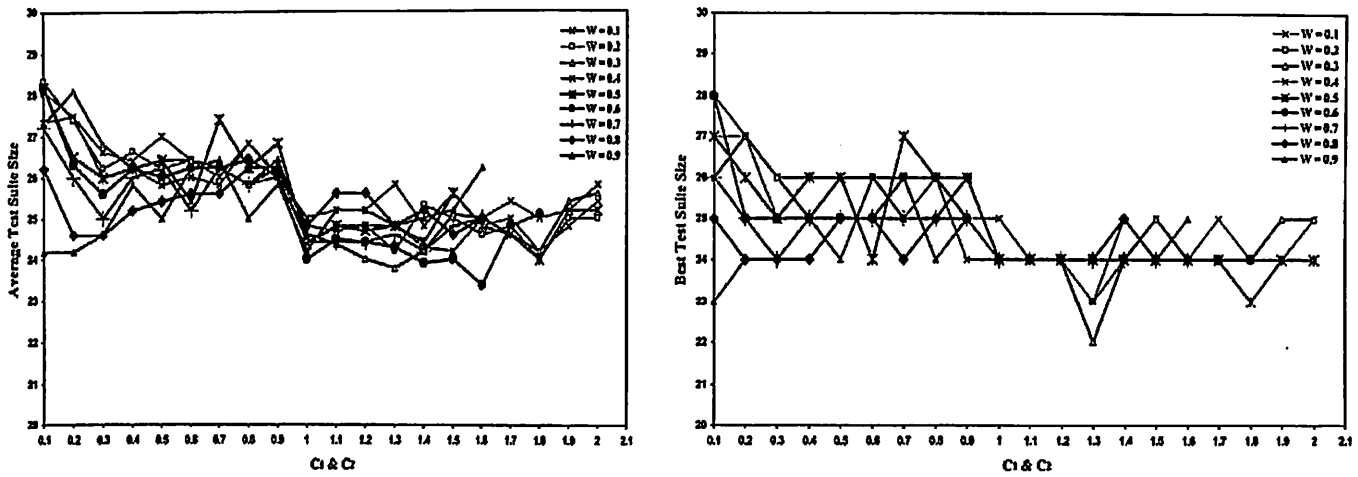


Figure 3-3 The Best and Average Sizes obtained with the Variation Inertia Component (w) and Acceleration Coefficients (c_1 , c_2) for CA (N; 2, 4^6)

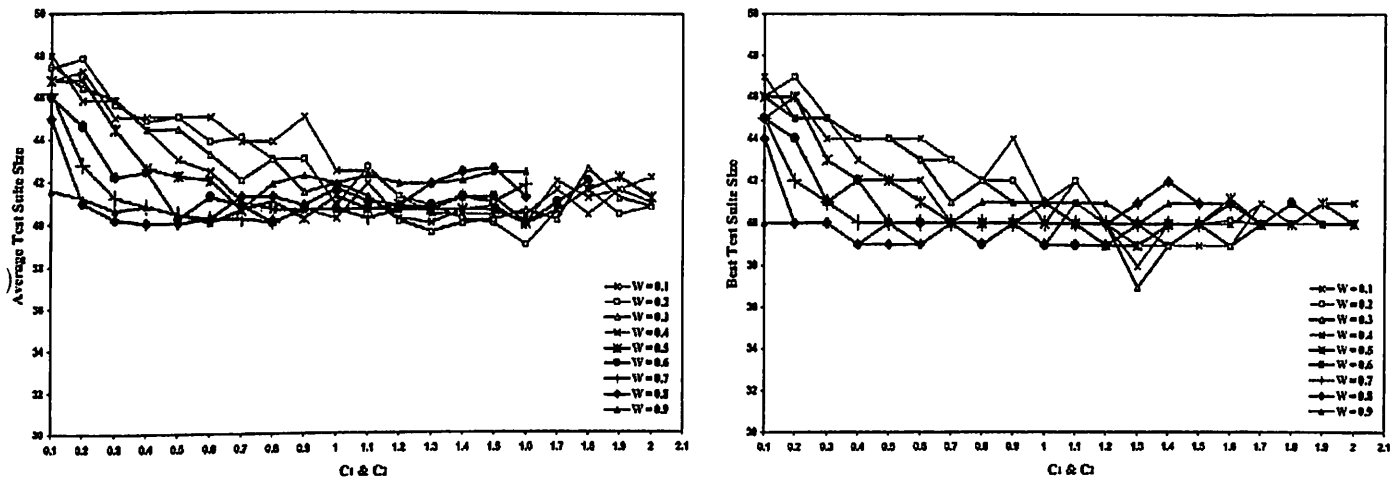


Figure 3-4 The Best and Average Sizes obtained with the Variation Inertia Component (w) and Acceleration Coefficients (c_1 , c_2) for CA (N; 2, 5^7)

leads the particle to move sharply towards the target region. Thus, there must be a balance in choosing these parameters.

In addition to the parameters mentioned above, the swarm search space and the iteration number also represent important parameters that must be considered. A large iteration number may consume more time without obtaining better solutions, whereas a small iteration number may lead to interruption of transition of the particles when they try to move through the best solutions. In the same way, a large swarm search space may consume more time, whereas a small search space may hinder the appearance of some good solutions.

According to [32, 61], PSO performs well when (c_1,c_2) are set between 0.5 and 2. Therefore, this range is used as the basis for the parameter-tuning process. The tuning process of (c_1, c_2) and w is performed with large values of repetition and the swarm size, i.e., 60 and 200, respectively, to ensure the appearance of the most optimal particles in the swarm search space. We have adopted the experiments undertaken by Stardom [56] for SA, GA, and TS parameter setting, while taking into account the parameter setting for PSO in the literature [61, 69, 70]. The experiments are performed to determine the optimal size of the CA by fixing c_1 and c_2 and by varying w, and then performing the reverse experiment. Here, to have a better statistical significance, each experiment is repeated 80 times and the best and average sizes are recorded. Two well-known CAs are used as the base problem for construction: CA (N; 2, 4^6), and CA (N; 2, 5^7), with

values deduced earlier [i.e., (c_1, c_2) =1.375 and w = 0.3]. Various values of the swarm size and repetition are then tested by fixing the swarm size and by varying the repetition, and then performing the reverse experiment. The average generation time is recorded in order to determine its effects with the variation of the swarm size and repetition. All experiments employ the two CAs used earlier, i.e., CA (N; 2, 4^6), and CA (N; 2, 5^7). Figures 3-5 and 3-6 depict the best and average test sizes obtained with the variation of the swarm size and repetition, whilst Figures 3-7 and 3-8 depict the average generation time obtained with the variation of the swarm size and repetition.

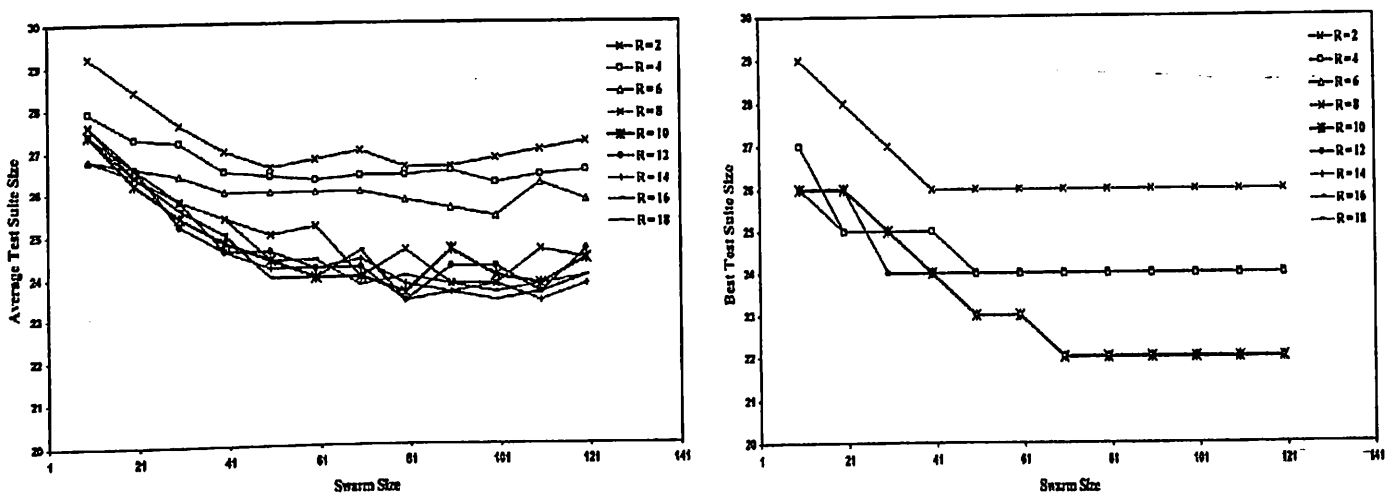


Figure 3-5 The Best and Average Sizes obtained with the Variation of Swarm Size and Repetition for CA (N; 2, 4^6)

As shown in Figures 3-3 and 3-4, the acceleration coefficients (c_1 and c_2) and their respective inertia weight (w) have a direct impact on the size obtained. The strategy generates poor results for the range of (c_1, c_2) greater than 0.1 and smaller than 1, whereas it performs well in case of (c_1, c_2) smaller than 1 when w increases towards 0.9. The interpretation of this observation is that the strategy seems to trust the global solution of (c_1, c_2) smaller than 1. Nevertheless, the strategy tends to facilitate local solutions, leading towards better choices of test cases, as w approaches 0.9.

However, because the range of w is greater than 0.7 and (c_1, c_2) are greater than 1.6, the strategy is either unable to generate any results, or it takes a long time to generate (non-optimal) results, which are not plotted in Figures 3-3 and 3-4. This observation implies that, with increasing w, the particle tends to go out of the search space faster, thereby hindering the appearance of some solutions. The strategy appears to be effective by forcing the particle to return it to the search space, thereby rendering inability, or taking a long time to reach the desired solution.

Based on the aforementioned discussion, a good choice for (c_1, c_2) would be a value greater than 1.2 and smaller than 1.4, whereas the suggested value of w would be between 0.2 and 0.5. Specifically, (c_1, c_2) =1.375 and w=0.3 give the best result.

After determining the suitable choices for w and (c_1, c_2), it is necessary to determine the swarm size and repetition. During the process, the values of c and w are fixed to the

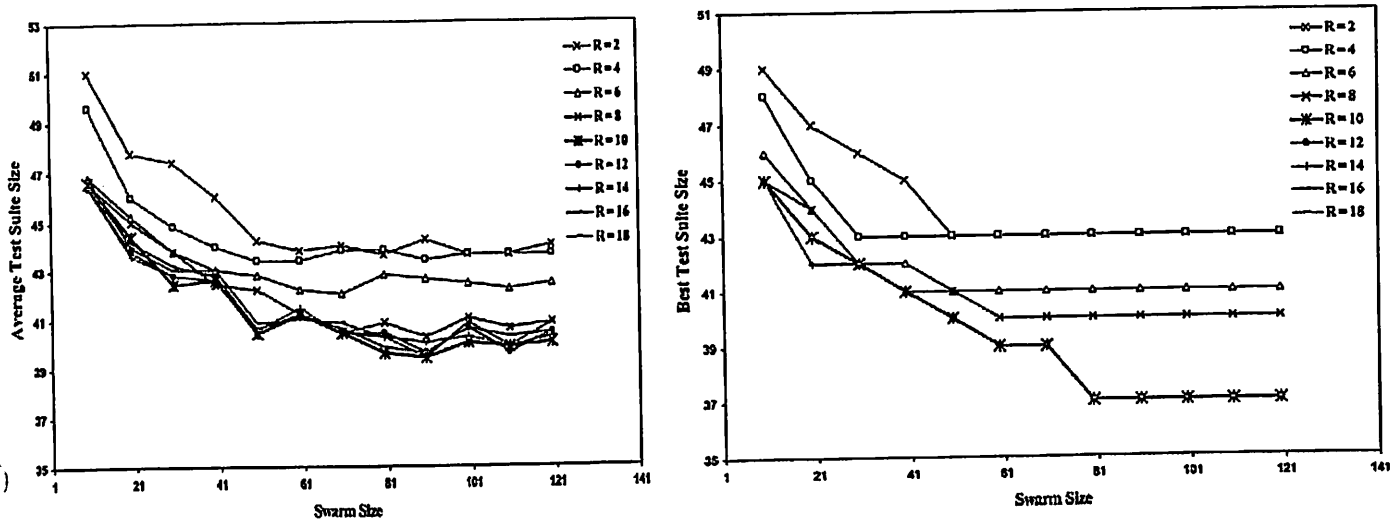


Figure 3-6 The Best and Average Sizes obtained with the Variation of Swarm Size and Repetition for *CA (N; 2, 5^7)*

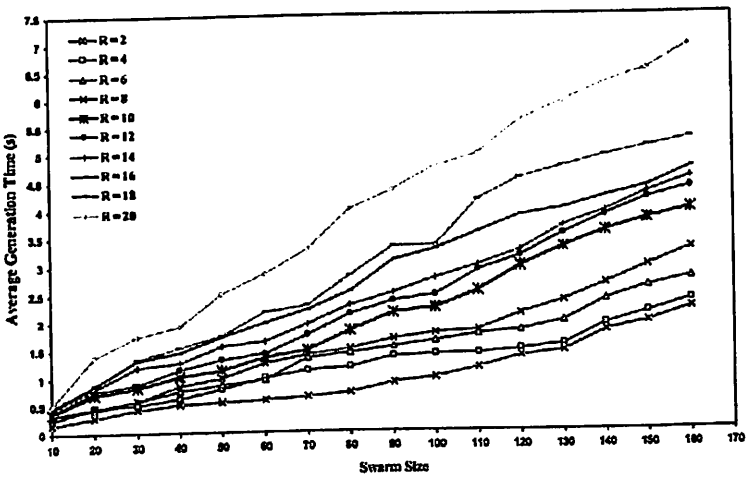


Figure 3-7 Average Generation Time obtained in second with the Variation of Swarm Size and Repetition for CA (N; 2, 4^6)

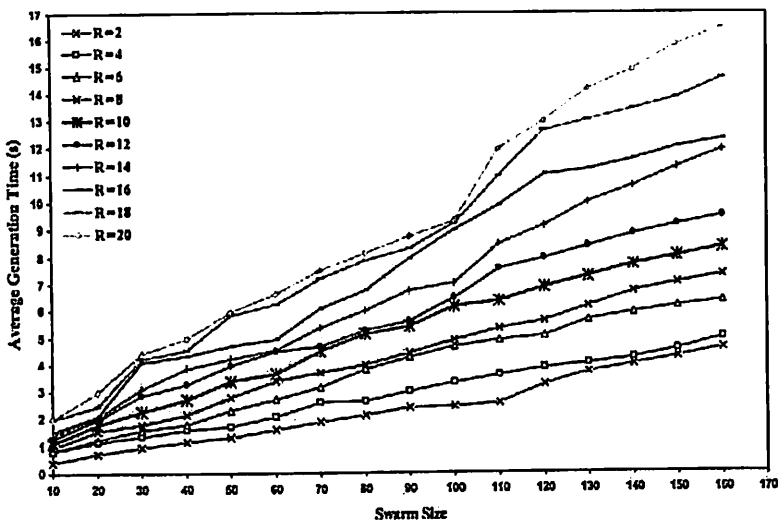


Figure 3-8 Average Generation Time obtained in second with the Variation of Swarm Size and Repetition for *CA (N; 2, 5^7)*

Figures 3-5 and 3-6 reveal that there is a trade-off between the swarm size and repetition. From the results, one can observe that, because of the larger search space, the increase in the swarm size leads to an improvement in the generated CA size. This improvement is not linear with the swarm size because sufficient repetitions are needed for the appearance of good solutions. However, it is noticeable from Figures 3-7 and 3-8 that the average generation time increases linearly with the swarm size and repetition.

From Figures 3-5 and 3-6, the best size is obtained from 80 repetitions. In the case of CA (N; 2, 4^6), the best size is obtained when the swarm size is 80. In contrast, in the case of CA (N; 2, 5^7), the best size is obtained when the swarm size is 90. After obtaining these values of iteration and swarm size, no significant improvements in the overall test sizes have been observed. This observation is because the result of the

strategy has arrived at the theoretical optimal sizes [19 in case of CA (N; 2, 4^6), and 29 in case of CA (N; 2, 5^7)]. There is a need to allow sufficient repetitions and swarm size to increase the chances of finding good (and optimal) sizes. For this reason, we have chosen 20 repetitions with a swarm size of 160 particles for all configurations. By choosing these values, we can also achieve a reasonable generation time, as shown in Figures 3-7 and 3-8.

3.4 Summary

The design and implementation of the PSTG strategy have been presented in this chapter. Specifically, the chapter presents the main algorithms of the strategy and illustrates how these algorithms are related to each other. The chapter also shows how PSO is used in the strategy to optimize the size of the generated test suite. In addition, the PSO parameter tuning process is also presented to ensure an optimal setting for better optimization of the strategy. The next chapter presents the experimental results of the evaluation process for the PSTG strategy.

CHAPTER 4

EVALUATION

Chapter 3 has illustrated the design and implementation of the PSTG strategy. Moreover, the design parameter setting of the strategy has been given in the chapter also. To characterize the performance of the strategy, this chapter presents the results of an extensive evaluation and characterization experimental process. The experiments are divided into three essential parts. In the first part, the *t*-way comparative experiments are performed to evaluate the proposed PSTG strategy in term of the generated test suite size. Then, the variable-strength experiments are carried out to compare against existing strategies. In both cases, the benchmarking process considers the existing AI- and computational-based strategies for comparison. The final part presents an empirical case study that was conducted on a non-trivial software system to show the applicability of the strategy and to determine the effectiveness of the generated test suites to detect faults.

4.1 Experimental Setup

For the first and the second parts of the experiments, the environment consists of a desktop PC with Windows 7, 2.8 GHz Core 2 Duo CPU, 3 GB of RAM, while the experimental environment for the third part consists of a a desktop PC with Ubuntu

10.10 operating system with gcc 4.4.5, 1.73 GHz Centrino Duo CPU, and 1 GB of RAM.

Throughout the experiments, each table represents the smallest test suite size obtained. The darkened cells with bold numbers show the best results obtained for the test configurations. Cells marked NA (Not Available) indicate that the results are unavailable for more than a week or it is not available from the literature, and cells marked NS (Not Supported) indicate that the tool is unable to generate the test case for a specific configuration.

All the AI-based strategies and some of the computational-based strategies produce non-deterministic results because they depend on some degree of randomness. The published results of those strategies were achieved by running each configuration 10 times and selecting the smaller size, which is considered the best test suite size (Cohen et al., 2003, Wang et al., 2008, Chen et al., 2009, Renee et al., 2007). However, for the other strategies, one run is sufficient because they do not depend on randomness and instead produce deterministic results. Since PSTG produces non-deterministic results, all configurations are executed 10 times to select the best test suite size.

4.2 The *t*-way Comparative Experiments

Here, basically the experiments are divided into two parts. The first part deals with evaluation and comparison of PSTG with existing published results from AI-based

strategies. As the generated test size is not influenced by the system specifications, the results are compared directly with those existing published results. The second part deals with comparison of PSTG with other computational-based strategies.

4.2.1 Comparing PSTG with Existing AI-based Strategies

At this stage of the experiments, the results are compared directly with those of SA, GA, and ACA, as published in (Shiba et al., 2004). In addition, the AETG and mAETG results are taken into account because both GA and ACA results are derived from them. The original results of AETG and mAETG are taken from (Cohen, 2004). Table 4-1 shows the sizes of the smallest test set generated by each strategy for $2 \leq t \leq 3$.

Table 4-1 Comparison with Existing AI-Based Strategies

Configurations	AETG	mAETG	GA	SA	ACA	PSTG
CA (N;2, 3^4)	9	9	9	9	9	9
CA (N;2, 3^{13})	15	17	17	16	17	17
CA (N;2, 2^{100})	NA	NA	12	NA	13	15
CA (N;2, 10^{10})	NA	NA	157	NA	159	163
CA (N;2, 10^{20})	180	198	227	183	225	229
CA (N;3, 5^6)	198	NA	125	152	125	167
CA (N;3, 3^6)	47	38	33	33	33	42
CA (N;3, 4^6)	105	77	64	64	64	102
CA (N;3, 10^6)	1508	1473	1501	1426	1496	1506
CA (N;3, 5^7)	229	218	218	201	218	229
CA (N;3, 6^6)	343	330	331	300	330	338
MCA (N;3, $10^1 6^2 4^3 3^1$)	NA	377	360	360	361	385
MCA (N;2, $5^1 4^4 3^{11} 2^5$)	30	25	26	21	25	28
MCA (N;2, $5^2 4^2 3^2$)	NA	114	108	100	106	112
MCA (N;2, $4^{15} 3^{17} 2^{29}$)	41	34	37	30	37	40
MCA (N;2, $5^1 3^8 2^2$)	19	20	15	15	16	21
MCA (N;2, $6^1 5^1 4^6 3^8 2^3$)	34	35	33	30	32	39
MCA (N;2, $7^1 6^1 5^1 4^6 3^8 2^3$)	45	44	42	42	42	48

In these configurations, PSTG performs poorly. Putting PSTG aside, if SA, GA, ACA, mAETG, and AETG are taken into account, clearly GA and ACA generate slightly better test sizes than AETG and mAETG for these configurations. Owing to its large random space, SA generates the best results in most cases.

The design and implementation of both GA and ACA mainly depend on the AETG algorithm. Here, GA and ACA are summoned after AETG. As illustrated in Chapter 2, the final test suite is further optimized by a "compaction algorithm" that merges the test cases for optimality. As a result, the underlying performances of both GA and ACA are not clear despite showing good results.

Concerning SA, despite producing the best overall result, to the best of our knowledge, there appears to be no reported SA results for $t>3$ in any configurations in the literature. For this reason, the performance of SA for high interaction strengths is unknown.

4.2.2 Comparing PSTG with Computational-based Strategies

To further demonstrate the performance in terms of test suites sizes, PSTG is compared with other well-known computational-based strategies, including Jenny (Jenkins, 2005), TConfig (Williams, 2008), ITCH (Hartman, 2005), PICT (Czerwonka, 2008), TVG (Arshem, 2009), CTX-XL (2003), and IPOG (Kuhn, 2009). All the tools are downloaded and implemented within the aforementioned environment. The comparison

aims to study the growth in the size of the test suites generated in terms of strength of coverage (t), the number of parameters (P), and the number of values (V).

To facilitate the comparison, different sets of experiments are established by adopting and extending the experiments conducted by (Lei et al., 2008, Lei et al., 2007, Calvagna and Gargantini, 2009) as well as by Bryce et al (2005). The experiments consist of seven sets of comparison, as follows:

i. **Experiment 1:** The number of parameters (P) and the parameters' values (V) are constant with the variation of the interaction strength (t) from 2 to 6.
ii. **Experiment 2:** The interaction strength (t) and the parameters' values (V) are set to 4 and 5 respectively with the variation of the parameter number (P) from 5 to 12.
iii. **Experiment 3:** The number of parameter (P) and the interaction strength (t) are set to 10 and 4 respectively, but the parameters' values (V) are varied from 2 to 6.
iv. **Experiment 4:** Five system configurations with mixed variable sizes with the constant interaction strength (t) set to 4.
v. **Experiment 5:** The number of parameters' values (V) is set to 3 and the interaction strength (t) is varied from 2 to 6, while the parameter number (P) is varied from 3 to 12 for each value of the interaction strength.

vi. **Experiment 6:** The number of parameters (P) is set to 7 and the interaction strength (t) is varied from 2 to 6, while the number of parameters' values (V) is varied from 2 to 5 for each value of the interaction strength.

vii. **Experiment 7:** Four real-world software system configurations are adopted.

The four real-world software system configurations that are adopted in the Experiment 7 are: Basic Billing System (BBS), Traffic Collision Avoidance System (TCAS), Mobile Phone, and Spin Simulator. BBS is a model of a basic telephone billing system consisting of four components. Each component has three values, as used in (Lott et al., 2005). TCAS represents the specification model of a software module part for 12 parameters (two 10-value parameters, one 4-value parameters, two 3-value parameters, and seven 2-value parameters), as presented in (Lei et al., 2007, Kuhn and Okum, 2006). In addition, the Mobile Phone example, as presented in (Cohen et al., 2007, Calvagna and Gargantini, 2009), models real-world mobile phone optional features in five parameters (two 2-value parameters, and three 3-value parameters). Finally, the Spin Simulator is a model-checking tool, as presented in (Holzmann, 1997, Calvagna and Gargantini, 2009). The tool is publicly available for use as a simulator undertaking the state machine run, or as a verifier used for specification properties check.

Tables 4-2 to 4-8 summarize the results for each stage of the above experiments respectively.

Table 4-2 P&V Constants (10, 5), But *t* Varied to 6

t-way	Jenny Size	TConfig Size	ITCH Size	PICT Size	TVG Size	CTE-XL Size	IPOG Size	PSTG Size
2	45	48	45	47	50	50	50	45
3	290	312	225	310	342	347	313	287
4	1719	1878	1750	1812	1971	NS	1965	1716
5	9437	NA	NS	9706	NA	NS	11009	9425
6	NA	NA	NS	47978	NA	NS	57290	50350

Table 4-3 *t*&V Constants (4, 5), But P Varied

P	Jenny Size	TConfig Size	ITCH Size	PICT Size	TVG Size	IPOG Size	PSTG Size
5	837	773	625	810	849	908	779
6	1074	1092	625	1072	1128	1239	1001
7	1248	1320	1750	1279	1384	1349	1209
8	1424	1532	1750	1468	1595	1792	1417
9	1578	1724	1750	1643	1795	1793	1570
10	1791	1878	1750	1812	1971	1965	1716
11	1839	2038	1750	1957	2122	2091	1902
12	1964	NA	1750	2103	2268	2285	2015

Table 4-4 P&*t* Constants (10, 4), But V Varied

V	Jenny	TConfig	ITCH	PICT	TVG	IPOG	PSTG
	Size	Size	Size	Size	Size	Size	Size
2	39	45	58	43	40	49	34
3	221	235	336	231	228	241	213
4	703	718	704	742	782	707	685
5	1719	1878	1750	1812	1917	1965	1716
6	3519	NA	NA	3735	4159	3935	3880

Table 4-5 Five Multi Domain Configurations

Configurations	Jenny	TConfig	ITCH	PICT	TVG	IPOG	PSTG
	Size	Size	Size	Size	Size	Size	Size
MCA (N;4, 3^4 4^5)	457	499	704	439	487	463	447
MCA (N;4, 5^1 3^8 2^2)	303	302	1683	310	313	324	292
MCA (N;4, 8^2 7^2 6^2 5^2)	4580	4317	4085	4565	5124	4776	4506
MCA (N;4, 6^5 5^4 3^2)	3033	NA	NA	2694	2881	3273	3154
MCA (N;4, 10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1)	6138	5495	5922	5916	6698	5492	5906

Table 4-6 Variable Number of Parameters $3 \leq P \leq 12$, Each With 3 Values t Varied to 6

t-way	P	Jenny Size	TConfig Size	ITCH Size	PICT Size	TVG Size	CTE-XL Size	IPOG Size	PSTG Size
2	3	9	10	9	10	10	10	11	9
	4	13	10	9	13	12	14	12	9
	5	14	14	15	13	13	14	14	12
	6	15	15	15	14	15	14	15	13
	7	16	15	15	16	15	16	17	15
	8	17	17	15	16	15	17	17	15
	9	18	17	15	17	15	18	17	17
	10	19	17	15	18	16	18	20	17
	11	17	20	15	18	16	20	20	17
	12	19	20	15	19	16	20	20	18
3	4	34	32	27	34	34	34	39	30
	5	40	40	45	43	41	43	43	39
	6	51	48	45	48	49	52	53	45
	7	51	55	45	51	55	54	57	50
	8	58	58	45	59	60	63	63	54
	9	62	64	75	63	64	66	65	58
	10	65	68	75	65	68	71	68	62
	11	65	72	75	70	69	76	76	64
	12	68	77	75	72	70	79	76	67
4	5	109	97	153	100	105	NS	115	96
	6	140	141	153	142	139		181	133
	7	169	166	216	168	172		185	155
	8	187	190	216	189	192		203	175
	9	206	213	306	211	215		238	195
	10	221	235	336	231	233		241	210
	11	236	258	348	249	250		272	222
	12	252	272	372	269	268		275	244
5	6	348	305	NS	310	321	NS	393	312
	7	458	477		452	462		608	441
	8	548	583		555	562		634	515
	9	633	684		637	660		771	598
	10	714	773		735	750		784	667
	11	791	858		822	833		980	747
	12	850	938		900	824		980	809
6	7	1087	921	NS	1015	1024	NS	1281	977
	8	1466	1515		1455	1484		2098	1402
	9	1840	1931		1818	1849		2160	1684
	10	2160	NA		2165	2192		2726	1980
	11	2459	NA		2496	2533		2739	2255
	12	2757	NA		2815	2597		3649	2528

Table 4-7 Seven Parameters, Each Having Variable Number of Values $2 \leq V \leq 5$, With t Varied to 6

t-way	V	Jenny	TConfig	ITCH	PICT	TVG	CTE-XL	IPOG	PSTG
		Size	Size	Size	Size	Size	Size	Size	Size
2	2	8	7	6	7	7	8	8	6
	3	16	15	15	16	15	16	17	15
	4	28	28	28	27	27	30	28	26
	5	37	40	45	40	42	42	42	37
3	2	14	16	13	15	15	15	19	13
	3	51	55	45	51	55	54	57	50
	4	124	112	112	124	134	135	208	116
	5	236	239	225	241	260	265	275	225
4	2	31	36	40	32	31	NS	48	29
	3	169	166	216	168	167		185	155
	4	517	568	704	529	559		509	487
	5	1248	1320	1750	1279	1385		1349	1176
5	2	57	56	NS	57	59	NS	128	53
	3	458	477		452	464		608	441
	4	1938	1792		1933	2010		2560	1826
	5	5895	NA		5814	6257		8091	5474
6	2	87	64	NS	72	78	NS	64	64
	3	1087	921		1015	1016		1281	977
	4	6127	NA		5847	5978		4096	5599
	5	23492	NA		22502	23218		28513	21595

Table 4-8 Four Real-World Software System Configurations, With t Varied to 6

t-way	Spec.	Task	Jenny	TConfig	ITCH	PICT	TVG	CTE-XL	IPOG	PSTG
			Size	Size	Size	Size	Size	Size	Size	Size
2	BBS	3^4	13	10	9	13	12	10	12	9
	TCAS	$2^7 3^2 4^1 10^2$	106	109	120	100	100	100	100	100
	Mobile Phone	$2^2 3^3$	12	12	15	10	10	9	11	9
	Spin Simulator	$2^{13} 4^5$	26	29	28	23	27	26	20	24
3	BBS	3^4	34	32	27	34	32	37	39	27
	TCAS	$2^7 3^2 4^1 10^2$	413	472	2388	400	434	426	400	400
	Mobile Phone	$2^2 3^3$	29	30	45	29	30	32	27	27
	Spin Simulator	$2^{13} 4^5$	111	113	196	96	111	113	78	101
4	TCAS	$2^7 3^2 4^1 10^2$	1536	1548	1484	1369	1599	NS	1377	1520
	Mobile Phone	$2^2 3^3$	59	56	138	59	55		54	54
	Spin Simulator	$2^{13} 4^5$	412	427	1296	353	288		341	380
5	TCAS	$2^7 3^2 4^1 10^2$	4621	NA	NS	4250	4773	NS	4283	4566
	Spin Simulator	$2^{13} 4^5$	1304	NA		1185	842		1243	1270
6	TCAS	$2^7 3^2 4^1 10^2$	11625	NA	NS	11342	NA	NS	11939	11743
	Spin Simulator	$2^{13} 4^5$	3538	NA		3420	NA		3516	3648

Table 4-2 shows that ITCH produces satisfactory results with small values of t (i.e., $t \leq 4$); however, no outputs are produced by ITCH after $t=4$ because it does not support the case $t>4$. Similar to ITCH, CTE-XL does not provide the support for $t>3$ but it can generate satisfactory result for $t=2$ and 3. TConfig and TVG provide the support for the case $t>4$ but also they do not produce any specific results in case of $t>4$ for these testing configurations. Jenny produces a reasonable result with $t=5$; however, it fails to produce any results with $t=6$. Although it does not generate any optimal solutions in all configurations, IPOG seems to be the only strategy that can generate test suite at $t=6$.

For the configuration in Table 4-3, the results for CTE-XL are not reported because it does not support $t>3$. ITCH produces the most optimal test sizes for P=5, P=6, P=11, and P=12. Jenny, PICT, TVG, and IPOG still produce satisfactory results but could not produce any optimal results for these configurations. It is noticeable that TConfig could not generate any specific results for P=12, although it generated reasonable results for the other parameters. PSTG produces the most optimal test sizes for P=7 to P=10.

Table 4-4 examines the sizes of the generated test suites by the strategies and tools with the values (V) growing. Here, ITCH generate mixed results, i.e., in some case, the generated test suite size is satisfactory compared with the other tools and strategies, while in some other cases, the generated test suite size is bigger than the others. In case of V=6, ITCH could not produce any specific result although it is support $t=4$. The strategies PICT, TVG, and TConfig also produced reasonable results but they could not produce any optimal results for these values. In addition, TConfig failed to produce the

test suite size in case of V=6. In another hand IPOG and Jenny produced better results than the aforementioned strategies in most cases. IPOG does not produce any optimal results, while Jenny produced the optimal test suite size in case of V=6. However, in most cases, PSTG produce optimal test suite sizes.

In the Tables 4-2 to 4-4, PSTG produces competitive results, and is able to compete with other strategies in most cases. Notably, the test sizes of PSTG grow exponentially and logarithmically as the number of *t*, parameters, and values increase in line with the theoretical value of O ($v^t \log p$) (Lei et al., 2007).

Considering multiple-domain (Mixed values) configurations in Table 4-5, PSTG appears to scale well against most other strategies in all configurations. In the case where PSTG is not the best, the test sizes are still within an acceptable value. The other strategies could generate optimal results for some of the configurations and failed to generate optimal sizes for some other configurations.

Based on the results obtained in Tables 4-6 to 4-8, CTE-XL generates satisfactory results in most cases. However, it is unable to generate competitive results in some cases. In addition, CTE-XL is unable to support values of *t* higher than 3. Similarly, TConfig, PICT, TVG, and Jenny produce satisfactory results in most cases. In fact, they produce optimal results for some cases. However, referring to certain test cases in the tables, these strategies failed to produce specific results when *t*=6. IPOG performs well for all values of *t*, although they do not generate the optimal solution in most cases.

However, IPOG seems to generate satisfactory results in cases of multi-domain configurations, as in the real-world systems in Table 4-8. ITCH generates satisfactory results for small values of *t*. However, it is unable to support values of *t* higher than 4.

Overall, the PSTG strategy outperforms other strategies in most of the configurations. In cases where the best results are not generated, the test suite size generated by PSTG is within an acceptable value and more competitive than most of the other strategies.

4.3 Variable-Strength Comparative Experiments

To benchmark against other variable-strength strategies, PSTG is compared with other available strategies that support the variable-strength test suite construction, including PICT, SA, ACS, TVG, Density, ParaOrder, ITCH, CTE-XL, and IPOG. Here also, the comparison aims to investigate the PSTG's generated test suite size against other strategies based on well-known benchmark configurations.

SA, ACS, Density, and ParaOrder strategies are not available for implementation and have little evidence to support them. Hence, results are directly compared with published results for strategies in Chen et al., Cohen et al., and Wang et al. (Cohen et al., 2003, Wang et al., 2008, Chen et al., 2009). The comparison is fair because the generated test size is not influenced by system specifications. However, publicly

available PICT, TVG, ITCH, CTE-XL, and IPOG strategies were implemented within the aforementioned environment.

The experiments consist of six sets of comparisons. For the first three sets, three available experiments conducted by Cohen (Cohen et al., 2003, Cohen, 2004) are adopted. In addition, the comparison results achieved by Wang et al. (2008) and Chen et al.(2009) for these experiments are included. However, as mentioned earlier, previous studies do not consider higher interactions. Thus, the experiments are extended to consider this situation also. For the other three experiments, three system configurations are adopted and several variable-strength settings are considered, particularly for high interaction strengths of t_m and t_s. In the first three sets of the experiments, most of the available and published strategies that support variable-strength test suite constructions were considered, i.e., ITCH, PICT, TVG, CTE-XL, ACS, SA, Density, ParaOrder, and IPOG. However, in the next three sets of the experiments, available strategies for implementation were merely considered, i.e., PICT, TVG, ITCH, and IPOG. The CTE-XL strategy is not considered also in the last three sets of the experiments because it does not support interaction strength higher than three. Tables 4-9 to 4-14 summarize the results obtained for these experiments.

Table 4-9 Sizes of Variable-Strength Interactional Test Suites for the Configuration VSCA (m; 2, 3^{15} , {C})

Configuration VSCA (m; 2, 3^{15} , a)										
{C}	ITCH	PICT	TVG	CTE-XL	ACS	SA	Density	ParaOrder	IPOG	PSTG
Ø	31	35	22	21	19	16	21	33	21	19
CA (3, 3^3)	48	81	27	28	27	27	28	27	27	27
CA (3, 3^3)^2	59	729	30	31	27	27	28	33	30	27
CA (3, 3^3)^3	69	785	30	31	27	27	28	33	33	27
CA (3, 3^4)	59	105	35	36	27	27	32	27	39	30
CA (3, 3^5)	62	131	41	43	38	38	40	45	39	38
CA (4, 3^4)	103	245	81	NS	NA	NA	NA	NA	81	81
CA (4, 3^5)	118	301	103	NS	NA	NA	NA	NA	122	97
CA (4, 3^7)	189	505	168	NS	NA	NA	NA	NA	181	158
CA (5, 3^5)	261	730	243	NS	NA	NA	NA	NA	243	243
CA (5, 3^7)	481	1356	462	NS	NA	NA	NA	NA	581	441
CA (6, 3^6)	745	2187	729	NS	NA	NA	NA	NA	729	729
CA (6, 3^7)	1050	3045	1028	NS	NA	NA	NA	NA	1196	966
CA (3, 3^4) CA (3, 3^5) CA (3, 3^6)	114	1376	53	49	40	34	46	44	51	45
CA (3, 3^6)	61	146	48	49	45	34	46	49	53	45
CA (3, 3^7)	68	154	54	55	48	41	53	54	58	49
CA (3, 3^9)	94	177	62	65	57	50	60	62	65	57
CA (3, 3^{15})	132	83	81	87	76	67	70	82	NS	74

Table 4-10 Sizes of Variable-Strength Interactional Test Suites for the Configuration VSCA (m; 2, $4^3\,5^3\,6^2$, {C})

Configuration VSCA (m; 2, $4^3\,5^3\,6^2$, {C})										
{C}	**ITCH**	**PICT**	**TVG**	**CTE-XL**	**ACS**	**SA**	**Density**	**ParaOrder**	**IPOG**	**PSTG**
Ø	48	43	44	47	41	**36**	41	49	43	42
CA (3, 4^3)	97	384	67	67	**64**	**64**	**64**	**64**	83	**64**
MCA (3, $4^3\,5^2$)	164	781	132	134	104	**100**	131	141	147	124
CA (3, 5^3)	145	750	**125**	**125**	**125**	**125**	**125**	126	136	**125**
MCA (4, $4^3\,5^1$)	354	1920	**320**	NS	NA	NA	NA	NA	329	**320**
MCA (5, $4^3\,5^2$)	1639	9600	**1600**	NS	NA	NA	NA	NA	1602	**1600**
CA (3, 4^3) **CA (3, 5^3)**	194	8000	**125**	126	**125**	**125**	**125**	129	136	**125**
MCA (4, $4^3\,5^1$) **MCA (4, $5^2\,6^2$)**	1220	288000	**900**	NS	NA	NA	NA	NA	**900**	**900**
CA (3, 4^3) **MCA (4, $5^3\,6^1$)**	819	48000	**750**	NS	NA	NA	NA	NA	**750**	**750**
CA (3, 4^3) **MCA (5, $5^3\,6^2$)**	4569	288000	**4500**	NS	NA	NA	NA	NA	**4500**	**4500**
MCA (4, $4^3\,5^2$)	510	2874	496	NS	NA	NA	NA	NA	512	**472**
MCA (5, $4^3\,5^3$)	2520	15048	2592	NS	NA	NA	NA	NA	2763	**2430**
MCA (3, $4^3\,5^3\,6^1$)	254	1266	237	236	201	**171**	207	247	215	206
MCA (3, $5^1\,6^2$)	188	900	**180**	**180**	**180**	**180**	**180**	**180**	**180**	**180**
MCA (3, $4^3\,5^3\,6^2$)	312	261	302	301	255	**214**	256	307	NS	260

Table 4-11 Sizes Of Variable-Strength Interactional Test Suites for the Configuration and VSCA (m; 2, $3^{20}10^2$, {C})

Configuration VSCA (m; 2, $3^{20}10^2$, {C})										
{C}	**ITCH**	**PICT**	**TVG**	**CTE-XL**	**ACS**	**SA**	**Density**	**ParaOrder**	**IPOG**	**PSTG**
Ø	NA	100	101	108	100	100	100	100	101	102
CA (3, 3^{20})	NA	940	103	121	100	100	100	103	100	105
MCA (3, 3^{20}, 10^2)	NA	423	423	509	396	304	401	442	NS	481
MCA (4, 3^3, 10^1)	NA	810	270	NS	NA	NA	NA	NA	273	270
MCA (5, 3^3, 10^2)	NA	2800	2700	NS	NA	NA	NA	NA	2700	2700
MCA (6, 3^4, 10^2)	NA	NA	8100	NS	NA	NA	NA	NA	8100	8100

Table 4-12 Test Size for Variable-Strength Configuration VSCA (m; 3, 3^{15} , {C})

Configuration VSCA (m; 3, 3^{15} , {C})					
{C}	ITCH	PICT	TVG	IPOG	PSTG
Ø	75	83	84	82	75
CA (4, 3^4)	129	1507	93	87	91
CA (4, 3^4)^2	183	19749	97	91	91
CA (4, 3^4)^3	237	531441	97	106	91
CA (4, 3^4)^3 CA(3, 3^3)	237	NA	98	106	90
CA (5, 3^5)	273	5366	244	248	248
CA (5, 3^5)^2	459	177300	245	250	245
CA (5, 3^5)^3	645	NA	245	261	245
CA (6, 3^6)	759	12609	729	729	729
CA (6, 3^6)^2	1431	NA	730	744	734
CA (6, 3^6)^2 CA(3, 3^3)	1431	NA	730	744	734
CA (4, 3^5)	151	1793	118	119	114
CA (5, 3^6)	287	5387	323	337	314
CA (6, 3^7)	1044	16792	1018	1215	969
CA (4, 3^7)	219	2781	168	183	159
CA (4, 3^9)	289	3095	214	227	195
CA (4, 3^{11})	354	2824	256	259	226
CA (4, 3^{15})	498	NS	327	NS	284
CA (5, 3^7)	481	7475	471	713	437
CA (5, 3^8)	620	8690	556	714	516
CA (5, 3^{10})	868	9774	745	862	665
CA (5, 3^{12})	NA	8909	925	1130	806
CA (5, 3^{15})	NA	NS	NA	NS	1024
CA (6, 3^8)	1513	22833	1479	2108	1396
CA (6, 3^9)	1964	26725	1840	2124	1690
CA (3, 3^4) CA (4, 3^5) CA (5, 3^6)	312	NA	331	419	312

Table 4-13 Test Size for Variable-Strength Configuration VSCA (m; 3, $4^1 3^7 2^2$, {C})

Configuration VSCA (m; 3, $4^1 3^7 2^2$, {C})					
{C}	ITCH	PICT	TVG	IPOG	PSTG
Ø	112	72	70	73	65
MCA (4, $4^1 3^3$)	193	1377	111	108	108
MCA (4, $4^1 3^3$) CA (4, 3^4)	253	17496	112	108	108
MCA (4, $4^1 3^4$)	217	1500	141	149	136
MCA (4, $4^1 3^5$)	226	1547	183	207	171
MCA (4, $4^1 3^4$) MCA (5, $3^3 2^2$)	307	NS	141	149	136
MCA (5, $4^1 3^4$) MCA (5, $3^3 2^2$)	482	NS	325	324	324

Table 4-14 Test Size for Variable-Strength Configuration VSCA (m; 2, $10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1${C})

Configuration VSCA (m; 2, $10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1${C})					
{C}	*ITCH*	*PICT*	*TVG*	*IPOG*	*PSTG*
Ø	119	102	99	91	97
MCA (3, $10^1 9^1 8^1$)	765	31256	720	720	720
MCA (3, $7^1 6^1 5^1$)	301	19515	210	221	210
MCA (3, $4^1 3^1 2^1$)	140	2397	99	91	97
MCA (3, $10^1 9^1 8^1 7^1$)	806	22878	784	772	742
MCA (3, $10^1 9^1 8^1$) MCA (3, $7^1 6^1 5^1$)	947	NA	720	720	720
MCA (3, $10^1 9^1 8^1$) MCA (6, $7^1 6^1 5^1 4^1 3^1 2^1$)	5803	NA	5040	5041	5040
MCA (3, $10^1 9^1 8^1$) MCA (3, $7^1 6^1 5^1$) MCA (3, $4^1 3^1 2^1$)	968	NA	720	720	720
MCA (4, $5^1 4^1 3^1 2^1$)	237	1200	123	142	120
MCA (5, $10^1 9^1 4^1 3^1 2^1$)	2276	124157	2160	2160	2160
MCA (6, $7^1 6^1 5^1 4^1 3^1 2^1$)	5157	NA	5040	5041	5040

Referring to the comparative results in Tables 4-9 to 4-11, when the strength of the interactions is small, such as in cases of small strengths (maximum two for t_m and three for t_s), SA frequently generates optimum results, and PICT generates the worst results. For the same configurations, ParaOrder, Density, TVG, and ITCH appear to generate satisfactory results most of the time. IPOG also generate reasonable results; however, the results are worse than those achieved by ParaOrder, albeit they use the same method of generation. PSTG and ACS generate results either equal or close to the results obtained by SA. Moreover, PSTG generates results similar to or more optimal than ACS.

Considering the configurations with interaction strengths higher than three, the results are unavailable for ParaOrder, Density, ACS, and SA. For the remaining implemented strategies, PICT still produces the worst results for most configurations, whereas PSTG produces the most optimal results. ITCH, TVG, and IPOG produce satisfactory results. TVG and IPOG frequently yield comparable results. PICT and IPOG are unable to produce results when the main and sub-configurations strength (t_m and t_s) are equal; as mentioned, such cases are marked NS in the tables.

In cases with higher strength configurations, as shown in Tables 4-12 to 4-14, merely the results of PICT, TVG, ITCH, and IPOG are reported, which are the only strategies that address this situation, to provide a comparison with PSTG. For these configurations, PICT still produces the worst results for most of the configurations, whereas PSTG

produces the most optimal results. TVG and IPOG produce satisfactory results; however, TVG generates results that are either similar to or comparable with those of PSTG in some cases.

4.4 Analysing the Results from *t*-way and Variable-Strength Experiments

In accordance with the *t*-way and variable-strength results, the strategies that generate satisfactory results in cases of *t*-way are not necessarily appropriate for the generation of variable-strength. In addition, finding a general strategy for the construction of both cases is difficult. For example, the results show that, although PICT generates reasonable results for *t*-way cases, it is not effective for variable-strength cases. The main reason is the construction algorithm wherein each parameter is independent, which leads to the loss of the ability to nest the parameters (Czerwonka, 2006). The problem of independence of the parameters can be solved by defining some negative values, such that a test case can only have one negative value appearing in a given test. However, this method is more restricted than nesting parameters because it is only applied to failure testing (Barrett and Dvorak, 2009). This, in turn, requires the construction method to generate more test cases, such as the construction of independent *t*-way test suite for the main and sub-configurations.

SA, on the other hand, often achieves small test sizes for most of the configurations, when the interaction strengths for the main and sub configurations are two and three, respectively. This optimality is owing to the use of the binary search process with a large random generated search space. Nevertheless, this construction method is time-consuming, particularly for interactions higher than three, or for more complicated configurations.

ACS also generates satisfactory results. In some cases, its results are similar or close to those from SA. However, in the implementation stage, the final test suite, which is the outcome of the algorithm, is optimized further by a merging algorithm that attempts to merge the test cases. As a result, the performance of the algorithms is not clear, and we cannot consider the outcome of the research as a pure performance of ACS, although research has shown impressive results for some small configurations when the interaction strengths for the main and sub-configurations are two and three, respectively. This elimination in interaction generation reveals that this method faces the problem of computation owing to the complex nature of the ACS algorithm.

ParaOrder and Density also generate satisfactory results. However, there is a lack of results for other configurations and interaction strengths. This gap has been filled for ParaOrder by the availability of IPOG, which uses the same construction method. IPOG

generates test suites more effectively, particularly for large-scale interaction strengths and configurations. However, its performance is similar to or worse than that of TVG, although its construction speed is better as compared with those of other strategies.

The TVG construction method is more effective, particularly for large-scale interaction strengths and configurations. However, for TVG and ITCH, the user must manually enter all interactions of the sub-configurations parameters, leading to a considerable loss of time in the entry procedure before the construction process.

PSTG is effective in configurations owing to its ability to locate solutions with a fewer number of moves. This effectiveness is also owing to the use of previous best-achieved test cases (*lBest*) in the update rule. This, in turn, leads to the generation of a search space surrounding the best solution, resulting in a set of better test cases. Therefore, the algorithm quickly locates the global best to be added to the final test suite.

4.5 Summary

This chapter highlighted the evaluation of the PSTG strategy extensively. Six different sets of experiments were performed to evaluate the efficiency of the chosen strategy and

were compared with existing strategies and tools. Overall, PSTG gives promising results. The next chapter summarises and concludes the findings and contributions of this research. The possible future directions of the research are also provided.

CHAPTER 5

CONCLUSION & FUTURE WORK

In this work, a novel interaction testing strategy, called PSTG, is implemented. The new strategy aimed to construct minimized *t*-way or variable-strength test suites by combining PSO with an algorithm for *t*-interaction element generation. The main focus of the work is to optimize the generated test suites sizes as well as to effectively detect interaction based faults.

In order to conclude and discuss the importance of the achieved results as well as the directions for the future researches, this chapter summarizes the earlier chapters' important indications and gives the future research directions.

5.1 Contribution

Summing up, a number of contributions can be derived from this research work. Firstly, this research work has investigated the use of PSO as the base of implementing a novel variable-strength and *t*-way strategy, PSTG. Furthermore, this work has also evaluated and benchmarked PSTG against both existing t-way and variable strength strategies. In

doing so, this work has also evaluated the effectiveness of PSTG for testing of real word applications. Based on the experimental benchmarks and case study evaluation in Chapter 4, a number of observations can be discussed here.

First, different kinds of CAs can be used as interaction test suites, which in turn used in the interaction testing process. As discussed previously in Chapter 1, the exhaustive testing is useful to tackle most of the faults in the software under test; however it is not applicable practically due to time, and resource constraints. Table 4-15 illustrates this situation clearly. From the table, it can be seen that the exhaustive test suite with 41,472 test cases can tackle most of the faults in the software. However, the same amount of faults can be tackled using MCA as a test suite with 764 test cases or using VSCA as a test suite with 104 test cases. Hence, the use of CAs as interaction test suites can serve as a compromise technique to exhaustive testing whilst complementing the other test suite design techniques.

Second, it can be seen from the results that interaction test suites construction (or the construction of CAs) is an optimization problem. Therefore optimization methods could be useful to be used within strategies for construction. Here, the prospect of using PSO for generating *t*-way and variable-strength test suites is useful. As such, using PSO here

with the proposed PSTG strategy offers a viable solution to undertake this problem, ranging from small to high interactions.

Third, as mentioned previously in Chapter 1, Tables 4-1 to 4-14 showed that searching for an optimum set of test cases can be a painstakingly difficult task, and it is a challenge to find a unified strategy that generates optimum results all the time. Therefore, it is customary to find a strategy that can achieve minimum test suites sizes for some interactions, parameters, or values, while it cannot be achieved by others.

Fourth, most of the strategies support the construction of CA and MCA (i.e., *t*-way test suites). However, few strategies support the construction of VSCA. A number of *t*-way strategies have started to support VSCA construction (e.g., PICT, IPOG, TVG, CTE-XL and SA). In addition to these strategies, a number of new strategies emerged to particularly construct VSCA using the published techniques. For those published strategies, there has been little evidence on their performance and how they construct the variable strength test suites.

Finally the limitation for existing strategies is going for higher *t*. As previously mentioned in Chapter 1, for practical concern, it is needed to support up to t=6, which is not provided by most of the implemented AI-based *t*-way strategies, as can be seen in

Table 4-1. In addition, for those strategies that support the generation of *t*-way test suites with interaction strength up to *t*=6, they may not necessarily support that situation in case of variable strength interaction. This situation can be seen clearly in Tables 4-12 to 4-14. Moreover, even those variable-strength strategies that support higher *t* (i.e., ITCH, PICT, TVG, and IPOG), few of them support the cumulative case, i.e., when the main and sub configurations strength are equals. Compared with existing strategies, the proposed PSTG strategy overcome those limitations and deal with the strength of interactions more flexibly than the existing strategies, that is, by supporting *t*-way and variable-strength interactions up to *t*=6.

5.2 Future Research Directions

There are different directions for this work in the future. Considering the current implementation of PSTG, introducing seeding and constraints is important for the future developments. In addition, releasing the beta version of the PSTG implementation is also a useful endeavour.

Considering the construction methods of the interaction test suites, this direction of research still needs further researches and investigations. There is a need to investigate more construction methods and identify better methods. In doing so, there is a distinct

possibility to produce new and better strategies by investigating the features of different optimization algorithms and combine them together to produce hybrid strategies.

As part of future work, exploring the applicability and effectiveness of the strategy on many real systems is also desirable. As can be noted from Chapter 2, there are few researches dealing with the application of interaction test suites. The strategy could be applicable in many research areas, such as hardware testing, by considering the interaction of the hardware components. It could also be effective in network performance evaluation by considering different interactions among the network components to achieve best performance.

Considering the application of interaction testing, different areas need more assessments and improvements, although the exiting researches achieved impressive results. One of those areas is the components' interaction testing. So far, the effectiveness of using the interaction test suites in this area is not clear. Although it is studied theoretically in many researches, there is little evidence showing its effectiveness. It seems to be encouraging area for an empirical study for the PSTG strategy. An interesting direction, for example, is to apply the strategy on e-commerce software systems. It is interesting if a research study the effect of the component interactions on some performance criteria practically. In addition, the application of VSCA is an open research direction also. It is noticeable from

the literature that there is little evidence for the application of VSCA. As illustrated in Chapter 2, this could be also applied with e-commence systems by taking stronger interaction strength among some special related components in the system, for example.

Finally, another important direction of research is to combine the interaction test suites features with other software testing methods. As previously mentioned, interaction test suites have been used with regression testing and test case prioritization. It also could be useful if the interaction test suite features are combined with fault localization techniques. So far, the use of interaction test suites has not been sufficiently investigated to address fault localization which in turn could further strengthen existing testing techniques.

REFERENCES

[1] W.C. Hetzel, B. Hetzel, The complete guide to software testing, John Wiley & Sons, Inc, 1991.

[2] M. Woodside, G. Franks, D.C. Petriu, The future of software performance engineering, in: Future of Software Engineering Conference, FOSE '07, IEEE Computer Society, Minneapolis, Minnesota, USA, 2007, pp. 171-187.

[3] E.J. Weyuker, F.I. Vokolos, Experience with performance testing of software systems: issues, an approach, and case study, IEEE Transactions on Software Engineering, 26 (2000) 1147-1156.

[4] D.S. Hoskins, C.J. Colbourn, D.C. Montgomery, Software performance testing using covering arrays: efficient screening designs with categorical factors, in: 5th International Workshop on Software and Performance, ACM, Palma, Illes Balears, Spain, 2005, pp. 131-136.

[5] F.T. Chan, T.Y. Chen, I.K. Mak, Y.T. Yu, Proportional sampling strategy: guidelines for software testing practitioners, Information and Software Technology, 38 (1996) 775-782.

[6] D.C. Montgomery, Design and analysis of experiments, John Wiley & Sons, 2006.

[7] T.J. Mitchell, An algorithm for the construction of "D-optimal" experimental designs, Technometrics, 42 (2000) 48-54.

[8] D. Hoskins, R.C. Turban, C.J. Colbourn, Experimental designs in software engineering: d-optimal designs and covering arrays, in: ACM Workshop on Interdisciplinary Software Engineering Research, ACM, Newport Beach, CA, USA, 2004, pp. 55 - 66.

[9] M. Chateauneuf, D.L. Kreher, On the state of strength-three covering arrays, Journal of Combinatorial Designs, 10 (2002) 217-238.

[10] X. Yuan, M.B. Cohen, A.M. Memon, GUI interaction testing: incorporating event context, IEEE Transactions on Software Engineering, 99 (2011).

[11] S. Huang, M.B. Cohen, A.M. Memon, Repairing GUI test suites using a genetic algorithm, in: 3rd IEEE International Conference on Software Testing, Verification and Validation, IEEE Computer Society, Washington, DC, USA, 2010, pp. 245-254.

[12] M. Ellims, D. Ince, M. Petre, The effectiveness of t-way test data generation, in: 27th international conference on Computer Safety, Reliability, and Security, Springer-Verlag, Newcastle upon Tyne, UK, 2008.

[13] C.J. Colbourn, Combinatorial aspects of covering arrays, Le Matematiche (Catania), 58 (2004) 121-167.

[14] M.C. Golumbic, I.B.-A. Hartman, A. Hartman, Software and hardware testing using combinatorial covering suites, in: R. Sharda, S. Voß (Eds.) Graph Theory, Combinatorics and Algorithms, Springer US, 2005, pp. 237-266.

[15] M.B. Cohen, P.B. Gibbons, W.B. Mugridge, C.J. Colbourn, J.S. Collofello, Variable strength interaction testing of components, in: 27th Annual International Conference on Computer Software and Applications, IEEE Computer Society, Dallas, Texas, USA, 2003, pp. 413 - 418.

[16] C. Yilmaz, M.B. Cohen, A. Porter, Covering arrays for efficient fault characterization in complex configuration spaces, ACM SIGSOFT Software Engineering Notes, 29 (2004) 45-54.

[17] W. Afzal, R. Torkar, R. Feldt, A systematic review of search-based testing for non-functional system properties, Information and Software Technology, 51 (2009) 957-976.

[18] G. Seroussi, N.H. Bshouty, Vector sets for exhaustive testing of logic circuits, IEEE Transactions on Information Theory, 34 (1988) 513-522.

[19] L. Yu, K.C. Tai, In-parameter-order: a test generation strategy for pairwise testing, in: 3rd IEEE International Symposium on High-Assurance Systems Engineering, IEEE Computer Society, Washington, DC , USA 1998, pp. 254-261.

[20] B. Jarboui, M. Cheikh, P. Siarry, A. Rebai, Combinatorial particle swarm optimization (CPSO) for partitional clustering problem, Applied Mathematics and Computation, 192 (2007) 337-345.

[21] S. Panda, N.P. Padhy, Comparison of particle swarm optimization and genetic algorithm for FACTS-based controller design, Applied Soft Computing, 8 (2008) 1418-1427.

[22] K.P. Anagnostopoulos, L. Kotsikas, Experimental evaluation of simulated annealing algorithms for the time-cost trade-off problem, Applied Mathematics and Computation, 217 (2010) 260-270.

[23] X. Chen, Q. Gu, A. Li, D. Chen, Variable strength interaction testing with an ant colony system approach, in: 16th Asia-Pacific Software Engineering Conference, IEEE Computer Society, Penang, Malaysia, 2009, pp. 160-167.

[24] Z. Wang, B. Xu, C. Nie, Greedy heuristic algorithms to generate variable strength combinatorial test suite, in: 8th International Conference on Quality Software, IEEE Computer Society, Oxford, UK, 2008, pp. 155-160.

[25] T. Shiba, T. Tsuchiya, T. Kikuno, Using artificial life techniques to generate test cases for combinatorial testing, in: 28th Annual International Computer Software and Applications Conference, IEEE Computer Society, Hong Kong, 2004, pp. 72-77 vol.71.

[26] D.R. Kuhn, M.J. Reilly, An investigation of the applicability of design of experiments to software testing, in: 27th Annual NASA Goddard Software Engineering Workshop (SEW-27'02), IEEE Computer Society, Greenbelt, Maryland, 2002, pp. 91.

[27] D.R. Kuhn, D.R. Wallace, A.M. Gallo, Jr., Software fault interactions and implications for software testing, IEEE Transactions on Software Engineering, 30 (2004) 418-421.

[28] S.R. Dalal, A. Jain, N. Karunanithi, J.M. Leaton, C.M. Lott, G.C. Patton, B.M. Horowitz, Model-based testing in practice, in: 21st International Conference on Software Engineering, ACM, Los Angeles, California, United States, 1999, pp. 285 - 294.

[29] J. Jie, J. Zeng, C. Han, Q. Wang, Knowledge-based cooperative particle swarm optimization, Applied Mathematics and Computation, 205 (2008) 861-873.

[30] M. Clerc, J. Kennedy, The particle swarm - explosion, stability, and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation, , 6 (2002) 58-73.

[31] M.P. Wachowiak, R. Smolikova, Z. Yufeng, J.M. Zurada, A.S. Elmaghraby, An approach to multimodal biomedical image registration utilizing particle swarm optimization, IEEE Transactions on Evolutionary Computation,, 8 (2004) 289-301.

[32] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Transactions on Evolutionary Computation, 10 (2006) 281 - 295.

[33] B.S. Ahmed, K.Z. Zamli, PSTG: a t-way strategy adopting particle swarm optimization, in: 4th Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation, IEEE Computer Society, Kota Kinabalu, Borneo, Malaysia, 2010, pp. 1-5.

[34] B.S. Ahmed, K.Z. Zamli, T-way test data generation strategy based on particle swarm optimization, in: 2nd International Conference on Computer Research and Development, IEEE Computer Society, Kuala Lumpur, Malaysia, 2010, pp. 93-97.

[35] B. Beizer, Software testing techniques (2nd ed.), Van Nostrand Reinhold Co., 1990.

[36] A.H. Ronneseth, C.J. Colbourn, Merging covering arrays and compressing multiple sequence alignments, Discrete Applied Mathematics, 157 (2009) 2177-2190.

[37] C.J. Colbourn, Strength two covering arrays: existence tables and projection, Discrete Mathematics, 308 (2008) 772-786.

[38] M.B. Cohen, Designing test suites for software interaction testing, in: Department of Computer Science, University of Auckland, 2004, pp. 185.

[39] C. Nie, H. Leung, A survey of combinatorial testing, ACM Computing Surveys, 43 (2011) 1-29.

[40] M. Grindal, J. Offutt, S.F. Andler, Combination testing strategies: a survey, Software Testing, Verification & Reliability, 15 (2005) 167 - 199.

[41] C.S. Cheng, Orthogonal arrays with variable numbers of symbols, The Annals of Statistics, 8 (1980) 447-453.

[42] Y. Lei, R. Kacker, D.R. Kuhn, V. Okun, J. Lawrence, IPOG-IPOG-D: efficient test generation for multi-way combinatorial testing, Software Testing, Verification & Reliability, 18 (2008) 125-148.

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

[44] J. Czerwonka, Pairwise testing in real world: practical extensions to test case generator, in: 24th Pacific Northwest Software Quality Conference, IEEE Computer Society, Portland, Oregon, USA, 2006, pp. 419-430.

[45] R.C. Bryce, C.J. Colbourn, The density algorithm for pairwise interaction testing: Research Articles, Software Testing, Verification & Reliability, 17 (2007) 159-182.

[46] R.C. Bryce, C.J. Colbourn, A density-based greedy algorithm for higher strength covering arrays, Software Testing, Verification & Reliability, 19 (2009) 37-53.

[47] E. Lehmann, J. Wegener, Test case design by means of the CTE XL, in: 8th European International Conference on Software Testing, Analysis & Review (EuroSTAR 2000), Kopenhagen, Denmark, 2000, pp. 1-10.

[48] Y.T. Yu, S.P. Ng, E.Y.K. Chan, Generating, selecting and prioritizing test cases from specifications with tool support, in: 3rd International Conference on Quality Software, IEEE Computer Society, Dallas, Texas, 2003, pp. 83-90.

[49] Y.-W. Tung, W.S. Aldiwan, Automating test case generation for the new generation of mission software system, in: IEEE Aerospace Conference, IEEE Computer Society, Big Sky, MT , USA 2000, pp. 431-437.

[50] j. Arshem, TVG download page, http://sourceforge.net/projects/tvg, 2009.

[51] B. Jenkins, Jenny download web page, in, Bob Jenkins' Web site, http://burtleburtle.net/bob/math/jenny.html, 2005

[52] A.W. Williams, Determination of test configurations for pair-wise interaction coverage, in: IFIP TC6/WG6.1 13th International Conference on Testing Communicating Systems: Tools and Techniques, Kluwer, B.V., Deventer, The Netherlands, 2000, pp. 59-74.

[53] A. Hartman, IBM Intelligent Test Case Handler, in, IBM alphaworks, http://www.alphaworks.ibm.com/tech/whitch, 2005.

[54] Y. Lei, R. Kacker, D.R. Kuhn, V. Okun, J. Lawrence, IPOG: a general strategy for t-way software testing, in: 4th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, IEEE Computer Society, Tucson, Arizona, 2007, pp. 549-556.

[55] A. Calvagna, A. Gargantini, IPO-s: incremental generation of combinatorial interaction test data based on symmetries of covering arrays, in: IEEE International Conference on Software Testing, Verification, and Validation Workshops, IEEE Computer Society, Denver, Colorado, USA, 2009, pp. 10-18.

[56] J. Stardom, Metaheuristics and the search for covering and packing array in: Department of Mathematics, Simon Fraser University, Canada, 2001, pp. 89.

[57] K.J. Nurmela, Upper bounds for covering arrays by tabu search, Discrete Applied Mathematics, 138 (2004) 143-152.

[58] B. Garvin, M. Cohen, M. Dwyer, Evaluating improvements to a meta-heuristic search for constrained interaction testing, Empirical Software Engineering, 16 (2011) 61-102.

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

[60] R. Kuhn, ACTS download page, 2011, in, National institute of standards and technology, information technology laboratory, 2009.

[61] A. Windisch, S. Wappler, J. Wegener, Applying particle swarm optimization to software testing, in: 9th Annual Conference on Genetic and Evolutionary Computation, ACM, London, England, 2007, pp. 1121-1128.

[62] R. Poli, Analysis of the publications on the applications of particle swarm optimisation, Journal of Artificial Evolution and Applications articles, 2008 (2008) 1-10.

[63] N.P. Padhy, Artificial intelligence and intellegent systems, Oxford University Press, 2009.

[64] Y. Marinakis, M. Marinaki, A hybrid multi-swarm particle swarm optimization algorithm for the probabilistic traveling salesman problem, Computers & Operations Research, 37 (2010) 432-442.

[65] J. Kennedy, R. Eberhart, Particle swarm optimization, in: IEEE International Conference Neural Networks, IEEE Computer Society, Perth, Australia, 1995, pp. 1942-1948.

[66] R.J. Kuo, L.M. Lin, Application of a hybrid of genetic algorithm and particle swarm optimization algorithm for order clustering, Decision Support Systems, 49 (2010) 451-462.

[67] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: 6th International Symposium on Micro Machine and Human Science, IEEE Computer Sociaty, Nagoya , Japan, 1995, pp. 39-43.

[68] A. Ganjali, A requirements-based partition testing framework using particle swarm optimization technique, in: Electrical and Computer Engineering, University of Waterloo, Ontario, Canada, 2008.

[69] Y. Shi, R.C. Eberhart, Empirical study of particle swarm optimization, in: IEEE Congress on Evolutionary Computation (CEC 1999), IEEE Computer Society, Piscataway, NJ, 1999, pp. 1945-1950.

[70] Z. Li-ping, Y. Huan-jun, a.H. Shang-xu, Optimal choice of parameters for particle swarm optimization, Journal of Zhejiang University - Science A, 6 (2005) 528-534.

[71] C.J. Colbourn, Covering array tables, in, Arizona State University.

[72] B. Stevens, Transversal Coverings and Packings, in: Graduate Department of Mathematics, University of Toronto, 1988, pp. 148.

[73] Renee, C. Bryce, C.J. Colbourn, One-test-at-a-time heuristic search for interaction test suites, in: 9th Annual Conference on Genetic and Evolutionary Computation, ACM, London, England, 2007, pp. 1082-1089.

[74] C. Lott, A. Jain, S. Dalal, Modeling requirements for combinatorial software testing, SIGSOFT Software Engineering Notes, 30 (2005) 1-7.

[75] D.R. Kuhn, V. Okum, Pseudo-Exhaustive testing for Software, in: 30th Annual IEEE/NASA Software Engineering Workshop, IEEE Computer Society, Columbia, Maryland, 2006, pp. 153-158.

[76] M.B. Cohen, M.B. Dwyer, J. Shi, Interaction testing of highly-configurable systems in the presence of constraints, in: International Symposium on Software Testing and Analysis, ACM, London, United Kingdom, 2007, pp. 129-139.

[77] G.J. Holzmann, The model checker SPIN, IEEE Transactions on Software Engineering, 23 (1997) 279-295.

[78] A. Barrett, D. Dvorak, A combinatorial test suite generator for gray-box testing, in: 3rd IEEE International Conference on Space Mission Challenges for Information Technology, IEEE Computer Society, Pasadena, California, USA, 2009, pp. 387-393.

[79] H. Do, S. Elbaum, G. Rothermel, Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact, Empirical Software Engineering: An International Journal, 10 (2005) 405-435.

[80] J.R. Ruthruff, S. Elbaum, G. Rothermel, Experimental program analysis, Information and Software Technology, 52 (2010) 359-379.

[81] E. Engstrm, P. Runeson, M. Skoglund, A systematic review on regression test selection techniques, Information and Software Technology, 52 (2010) 14-30.

[82] Count lines of code web page, in.

[83] J.H. Andrews, L.C. Briand, Y. Labiche, Is mutation an appropriate tool for testing experiments?, in: 27th international conference on Software engineering, ACM, St. Louis, MO, USA, 2005, pp. 402-411.

[84] C. Liu, L. Fei, X. Yan, J. Han, S.P. Midkiff, Statistical debugging: a hypothesis testing-based approach, IEEE Transactions on Software Engineering, 32 (2006) 831-848.