ELA: AN AUTOMATED STATISTICAL FAULT LOCALIZATION TECHNIQUE

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY
ÖZKAN BAYRAKTAR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

JUNE 2015

ELA: AN AUTOMATED STATISTICAL
FAULT LOCALIZATION TECHNIQUE

Submitted by **Özkan BAYRAKTAR** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Information Systems**, **Middle East Technical University** by,

Prof. Dr. Nazife Baykal _____
Director, **Informatics Institute**

Prof. Dr. Yasemin Yardımcı Çetin _____
Head of Department, **Information Systems**

Assoc. Prof. Dr. Aysu Betin Can _____
Supervisor, **Information Systems**

Examining Committee Members:

Prof. Dr. Nazife Baykal _____
Information Systems, METU

Assoc. Prof. Dr. Aysu Betin Can _____
Information Systems, METU

Assoc. Prof. Dr. Pınar Karagöz _____
Computer Engineering, METU

Assist. Prof. Dr. Tuğba Taşkaya Temizel _____
Information Systems, METU

Assoc. Prof. Dr. Vahid Garousi Yusifoğlu _____
Computer Engineering, Hacettepe University

Date: **18/06/2015**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Özkan Bayraktar

Signature :

ABSTRACT

ELA: AN AUTOMATED STATISTICAL FAULT LOCALIZATION TECHNIQUE

BAYRAKTAR, Özkan
Ph. D., Department of Information Systems
Supervisor: Assoc. Prof. Dr. Aysu Betin Can

June 2015, 135 pages

Software debugging consists of locating software faults, finding their causes, and fixing them. Among all these activities, the fault localization is the most difficult one and requires manual effort. Although there are several studies on automating this process, their effectiveness has not yet reached at a desired level.

In this dissertation, we propose a fault localization framework that introduces a new fault localization metric called Ela, three test suite reduction strategies to improve the effectiveness of fault localization, and an effective ranking strategy to improve the ranking of statements. Several experiments are performed on the Siemens suite to evaluate the proposed metric. Besides the expense metric used in fault localization literature, we also adapt the mean reciprocal rank to measure the overall ranking quality of the four techniques. Ela has better ranking than the other techniques in 4 of 118 versions while it is one of the best performing techniques for the remaining 114 versions of the subject programs.

We apply an equivalent test elimination strategy to neutralize the bias caused by the existence of the equivalent tests. This strategy achieves on average 99.5% test size reduction. Ela has better ranking than the other techniques in 31 of 118 versions while it is one of the best performing techniques for the remaining 87 versions of the subject programs.

We propose three test suite reduction strategies to reduce the effort for the fault localization. The best of these strategies achieves on average 34.1% test size reduction while resulting an improvement up to 1.7 in Jaccard, up to 2.46% in Tarantula, up to 1.01% in Ochiai, and up to 0.38% in Ela in terms of average expense.

We propose an effective ranking strategy, called Local Maxima, to improve the ranking of statements. This strategy achieves an improvement 10.54% in Jaccard, 10.47% in Tarantula, 10.74% in Ochiai, and 10.88% in Ela in terms of average expense.

Keywords: Software Testing, Statement Coverage, Statistical Fault Localization, Test Suite Reduction, Local Maxima

ÖZ

ELA: OTOMATİK İSTATİSTİKSEL HATA YERELLEŞTİRME TEKNİĞİ

BAYRAKTAR, Özkan
Doktora, Bilişim Sistemleri Bölümü
Danışman: Doç. Dr. Aysu Betin Can

Haziran 2015, 135 sayfa

Yazılım hatalarını ayıklama yazılım hatalarının yerelleştirilmesi, hataların nedenlerinin bulunması, ve hataların düzeltilmesinin kapsamaktadır. Bu aktivitiler içerisinde, hata yerelleştirme en zor aktivitedir ve elle düzeltme eforu gerektirmektedir. Bu işlemin otomatikleştirilmesi konusunda yapılan farklı çalışmalar olmasına rağmen, bu çalışmaların etkisi hala istenen seviyeye ulaşmamıştır.

Bu doktora tezinde, Ela olarak adlandırılan yeni bir hata yerelleştirme metriği, hata yerelleştirme etkinliğini iyileştirmek için üç adet test suit küçültme stratejisi, ifade sıralamayı iyileştirmek için bir etkin sıralama stratejisi içeren bir hata yerelleştirme çerçevesi önerdik. Önerilen metriği ölçmek için Siemens suiti üzerinde deneyler yapıldı. Literatürde kullanılan gider metriğinin yanı sıra dört tekniğin kalitesini ölçmek için ortalama ters sıra metriği de adapte edildi. Söz konusu programların 118 versiyonunun 4'ünde Ela tekniği diğer tekniklerden daha iyi sıralama yeteneğine sahipken kalan 114 versiyondaysa en iyi sıralama yeteneğine sahip tekniklerden birisidir.

Eşdeğer testlerin varlığının ifade sıralamada neden olduğu sapmayı etkisizleştirmek için eşdeğer test eleme stratejisi uyguladık. Bu strateji ortalama 99.5% test suit küçültme elde etmektedir. Söz konusu programların 118 versiyonunun 31'inde Ela tekniği diğer tekniklerden daha iyi sıralama yeteneğine sahipken kalan 87 versiyondaysa en iyi sıralama yeteneğine sahip tekniklerden birisidir.

Hata yerelleştirme eforunu düşürmek için üç adet test suit küçültme stratejisi önerdik. Bu stratejilerin en iyisi ortalama %34.1 test suit küçültme sağlarken, ortlama gider açısından Jaccard'ta %1.7'e kadar, Tarantula'da %2.46'ya kadar, Ochiai'de %1.01'e kadar, ve Ela'da %0.38'e kadar iyileşme elde etmektedir.

İfade sıralamayı iyileştirmek için Yerel Maksimum olarak adlandırılan bir etkin sıralama stratejisi önerdik. Bu strateji ortalama gider açısından Jaccard'ta %10.54, Tarantula'da %10.47, Ochiai'de %10.74, ve Ela'da %10.88 iyileşme elde etmektedir.

Anahtar Kelimeler: Yazılım Testi, İfade Kapsama, İstatistiksel Hata Yerelleştirme, Test Suit Küçültme, Yerel Maksimum

To my family: Pelin Bayraktar and Ela Nil Bayraktar

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my supervisor, Assoc. Prof. Dr. Aysu Betin Can for her support, guidance, and friendly encouragement. I appreciate her contributions of time and ideas to make my Ph.D. experience productive and stimulating.

I am very grateful to Assoc. Prof. Dr. Pınar Karagöz, Assist Prof. Dr. Tuğba Taşkaya Temizel, Prof. Dr. Nazife Baykal, and Assoc. Prof. Dr. Vahid Garousi Yusifoğlu for their valuable suggestions and comments.

I would like to thank my dear friend H. Arda Nural for his encouragement throughout the research.

I would like to express my sincere appreciation to my father Dursun Bayraktar, my mother Safiye Bayraktar, and my mother-in-law Müfide Beriat for their constant love and endless support.

Very special thanks go to my wife Pelin Bayraktar for her endless support, encouragement, and suggestions during all the hard times since the beginning of the research. Finally, my kisses go to my daughter: Ela Nil Bayraktar.

TABLE OF CONTENTS

ABSTRACT iv
ÖZ v
DEDICATION vi
ACKNOWLEDGEMENTS vii
TABLE OF CONTENTS viii
LIST OF TABLES xii
LIST OF FIGURES xv
LIST OF ABBREVIATIONS xvii
CHAPTERS
1. INTRODUCTION 1
1.1. Research Questions 5
1.2. Contributions of the Dissertation 6
1.3. Structure of the Dissertation 6
2. LITERATURE REVIEW 7
2.1. Fault Localization 7
2.1.1. Traditional Debugging Technique 7
2.1.2. Algorithmic Debugging Technique 8
2.1.3. Program Slice Based Technique 8
2.1.3.1. Set Difference Technique (SD) 8
2.1.3.2. Set Union (SU) & Set Intersection (SI) Techniques 8
2.1.3.3. Nearest Neighbor (NN) Technique 9
2.1.4. Memory State Based Technique 9
2.1.5. Statistical Technique 9
2.1.5.1. Predicate Based Technique 10
2.1.5.2. Statement Based Technique 11
2.1.5.3. Method Based Technique 12
2.1.5.4. Block Based Technique 12
2.2. Test Suite Reduction and Test Suite Selection 13
2.3. Surveys about Fault Localization, Test Suite Reduction and Test Suite Selection 17
3. PRELIMINARIES 19
3.1. Suspiciousness Metrics 21
3.2. Evaluation Metrics 22
3.2.1. Expense 22
3.2.2. Code Examination Effort 22

3.3. Mean Reciprocal Rank (MRR) 23
3.4. Subject programs 23
4. PROPOSED TECHNIQUE: ELA 25
4.1. Methodology 25
4.1.1. The Effects of the Bases of $A_{ef}(s_i)/A_f$ and $A_{np}(s_i)/A_p$ 26
4.2. Motivating Examples 30
4.3. Discussion on the Application of Ela to the Multiple Faults 34
5. PROPOSED TEST SUITE REDUCTION STRATEGIES 37
5.1. Heuristic I: FminCov Test Suite Strategy 37
5.2. Heuristic II: FminCov Cluster Test Suite Strategy 38
5.2.1. Visualization of tests via Multi-Dimensional Scaling (MDS) 41
5.3. Heuristic III: FminCov Classify Test Suite Strategy 46
5.3.1. Visualization of tests via Multi-Dimensional Scaling (MDS) 47
5.4. Heuristic IV: Equivalent Test Elimination (Distinct Test Selection) 50
5.4.1. Discussion 53
6. EFFECTIVE RANKING STRATEGY: LOCAL MAXIMA 55
6.1. Methodology 55
6.2. Motivating Example 57
6.3. Effect of LM on PrintTokens program 61
7. EXPERIMENTAL EVALUATION 63
7.1. Subject Programs 63
7.2. Experimental Results I: Comparison with the Three Prominent Fault Localization Techniques on the Original Test Suite 64
7.2.1. Significance Analysis of Ela Effectiveness on Redundant Test Suite 68
7.3. Experiment II: Test Reduction Strategy I – Distinct Test Suite 68
7.3.1. Experimental Results II: Comparison with the Three Prominent Fault Localization Techniques with Distinct Test Suite 69
7.3.2. Significance Analysis of Ela Effectiveness on Distinct Test Suite 72
7.4. Experiment III: Test Reduction Strategy II – Distinct FminCov Test Suite 73
7.4.1. Experimental Results III: Comparison with the Three Prominent Fault Localization Techniques with Distinct FminCov Test Suite 73
7.4.2. Significance Analysis of Ela Effectiveness on Distinct FminCov Test Suite 76
7.4.3. Significance Analysis of Failed Test Reduction 76
7.5. Experiment IV: Test Reduction Strategy III - Distinct FminCov Cluster Test Suite 76
7.5.1. Experimental Results IV: Comparison with the Three Prominent Fault Localization Techniques with Distinct FminCov Cluster Test Suite 77
7.5.2. Significance Analysis of Ela Effectiveness on Distinct FminCov Cluster Test Suite 80
7.5.3. Significance Analysis of Passed Test Reduction with Clustering 80

7.6. Experiment V: Test Reduction Strategy IV - Distinct FminCov Classify Test Suite 80
7.6.1. Experimental Results V: Comparison with the Three Prominent Fault Localization Techniques with Distinct FminCov Classify Test Suite 80
7.6.2. Significance Analysis of Ela Effectiveness on Distinct FminCov Classify Test Suite 84
7.6.3. Significance Analysis of Passed Test Reduction with Classification 84
7.7. Experiment VI: Effective Ranking Strategy: Local Maxima 84
7.7.1. Effective Ranking Strategy on Redundant Test Suite 84
7.7.1.1. Significance Analysis of Local Maxima with Ela Technique on Redundant Test Suite 87
7.7.2. Effective Ranking Strategy on Distinct Test Suite 87
7.7.2.1. Significance Analysis of Local Maxima with Ela Technique on Distinct Test Suite 90
7.7.3. Effective Ranking Strategy on Distinct FminCov Test Suite 90
7.7.3.1. Significance Analysis of Local Maxima with Ela Technique on Distinct FminCov Test Suite 93
7.7.4. Effective Ranking Strategy on Distinct FminCov Cluster Test Suite 93
7.7.4.1. Significance Analysis of Local Maxima with Ela Technique on Distinct FminCov Cluster Test Suite 96
7.7.5. Effective Ranking Strategy on Distinct FminCov Classify Test Suite 96
7.7.5.1. Significance Analysis of Local Maxima with Ela Technique on Distinct FminCov Classify Test Suite 99
7.7.6. Discussion on the Results 99
8. THREATS TO VALIDITY 101
8.1. Internal Validity 101
8.2. External Validity 101
8.3. Construct Validity 102
8.4. Conclusion Validity 102
9. CONCLUSIONS 103
10. FUTURE WORK 107
10.1. Application of Ela to Different Test Suites from Different Scales 107
10.2. Application of Ela with Different Coverage Entities 107
10.3. Investigating Optimal Percentage of Failed Tests in Test Suites 108
10.4. Application of Ela to Multiple Faults 108
10.5. Designing User Studies on Focus Groups 108
REFERENCES 109
APPENDICES
A. WILCOXON SIGNED RANK TESTS 117
B. FAULT TYPES AND THEIR DIFFICUILTIES 133

CURRICULUM VITAE 135

LIST OF TABLES

Table 1 – Metrics used in the suspiciousness formulas of the fault localization techniques.. 21
Table 2 – The seven C programs in SIR..... 24
Table 3 – Five passed test cases and one failed test case written for mid() function 30
Table 4 – Coverage matrix and result vector of mid() function 31
Table 5 – The suspiciousness values and ranks of the statements on mid() function 31
Table 6 – Coverage matrix and result vector of unblock_process function 33
Table 7 – The suspiciousness values and ranks of the statements on unblock_process function..... 34
Table 8 – The Cophenetic Correlation Coefficients for different linkage criteria when Hamming distance is used as a distance metric..... 40
Table 9 – Hamming distances between tests on PrintTokens_v01 with Distinct FminCov Cluster Test Suite 41
Table 10 – Hamming distances between tests on PrintTokens_v01 with Distinct FminCov Classify Test Suite..... 48
Table 11 – Improvements on Redundant Test Suite where FailMinCov tests are replicated by 2..... 51
Table 12 – Declines on Redundant Test Suite where FailOther tests are replicated by 2 52
Table 13 – Improvement of Local Maxima Strategy for Expenses on PrintTokens_v01 with Distinct Test Suite 62
Table 14 – The seven C programs in SIR..... 64
Table 15 – The seven C programs in SIR after equivalent test elimination 64
Table 16 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Redundant Test Suite..... 66
Table 17 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Redundant Test Suite..... 67
Table 18 – Code examination efforts of four techniques on the subject programs 67
Table 19 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct Test Suite 70
Table 20 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct Test Suite 71
Table 21 – Code examination efforts of four techniques on the subject programs 71
Table 22 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Test Suite..... 74
Table 23 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Test Suite..... 75
Table 24 – Code examination efforts of four techniques on the subject programs 75
Table 25 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Cluster Test Suite 78

Table 26 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Cluster Test Suite 79
Table 27 – Code examination efforts of four techniques on the subject programs 79
Table 28 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Classify Test Suite 82
Table 29 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Classify Test Suite 83
Table 30 – Code examination efforts of four techniques on the subject programs 83
Table 31 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Redundant Test Suite 85
Table 32 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Redundant Test Suite 86
Table 33 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct Test Suite 88
Table 34 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct Test Suite 89
Table 35 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct FminCov Test Suite 91
Table 36 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Test Suite 92
Table 37 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct FminCov Cluster Test Suite 94
Table 38 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Cluster Test Suite 95
Table 39 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct FminCov Classify Test Suite 97
Table 40 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Classify Test Suite 98
Table 41 – Wilcoxon signed rank test for the significance test of Ela and Jaccard on seven programs with Redundant Test Suite 117
Table 42 – Wilcoxon signed rank test for the significance test of Ela and Tarantula on seven programs with Redundant Test Suite 118
Table 43 – Wilcoxon signed rank test for the significance test of Ela and Ochiai on seven programs with Redundant Test Suite 119
Table 44 – Wilcoxon signed rank test for the significance test of Ela and Jaccard on seven programs with Distinct Test Suite 120
Table 45 – Wilcoxon signed rank test for the significance test of Ela and Tarantula on seven programs with Distinct Test Suite 121
Table 46 – Wilcoxon signed rank test for the significance test of Ela and Ochiai on seven programs with Distinct Test Suite 122
Table 47 – Wilcoxon signed rank test for the significance test of Ela and Jaccard on seven programs with Distinct FminCov Test Suite 123

Table 48 – Wilcoxon signed rank test for the significance test of Ela and Tarantula on seven programs with Distinct FminCov Test Suite 124
Table 49 – Wilcoxon signed rank test for the significance test of Ela and Ochiai on seven programs with Distinct FminCov Test Suite 125
Table 50 – Wilcoxon signed rank test for the significance test of Ela and Jaccard on seven programs with Distinct FminCov Cluster Test Suite 126
Table 51 – Wilcoxon signed rank test for the significance test of Ela and Tarantula on seven programs with Distinct FminCov Cluster Test Suite 127
Table 52 – Wilcoxon signed rank test for the significance test of Ela and Ochiai on seven programs with Distinct FminCov Cluster Test Suite 128
Table 53 – Wilcoxon signed rank test for the significance test of Ela and Jaccard on seven programs with Distinct FminCov Classify Test Suite 129
Table 54 – Wilcoxon signed rank test for the significance test of Ela and Tarantula on seven programs with Distinct FminCov Classify Test Suite 130
Table 55 – Wilcoxon signed rank test for the significance test of Ela and Ochiai on seven programs with Distinct FminCov Classify Test Suite 131
Table 56 – Fault types and their difficulties in seven programs of Siemens test suite 133

LIST OF FIGURES

Figure 1 – Coverage matrix and result vector 20

Figure 2 – Effect of the base of A_{ef}/A_f from 2 to 10 27

Figure 3 – Effect of the base of A_{np}/A_p from 2 to 10 28

Figure 4 – Effect of the base of A_{np}/A_p between 1 and 2 29

Figure 5 – Source code of mid() function 30

Figure 6 – Source code of unblock_process function 32

Figure 7 – Example code segment and its coverage matrix 37

Figure 8 – Failed and pass tests before clustering of failed and passed tests on PrintTokens_v01 42

Figure 9 – Silhouette coefficients for 2 clusters on PrintTokens_v01 42

Figure 10 – Silhouette coefficients for 3 clusters on PrintTokens_v01 43

Figure 11 – Silhouette coefficients for 4 clusters on PrintTokens_v01 43

Figure 12 – Silhouette coefficients for 5 clusters on PrintTokens_v01 44

Figure 13 – Selection of optimum ‘k’ according to the mean silhouette values 44

Figure 14 – Failed and pass tests after clustering of failed and passed tests on PrintTokens_v01 45

Figure 15 – Dendrogram of tests on PrintTokens_v01 [Clustering] 46

Figure 16 – The effect of FminCov Cluster Test Suite Strategy on PrintTokens_v01 46

Figure 17 – Failed and pass tests before classifying the passed tests to the failed tests on PrintTokens_v01 48

Figure 18 – Failed and pass tests after classifying the passed tests to the failed tests on PrintTokens_v01 49

Figure 19 – Dendrogram of tests on PrintTokens_v01 [Classification] 49

Figure 20 – The effect of FminCov Classify Test Suite Strategy on PrintTokens_v01 50

Figure 21 – Improvements by replicating the FailMinCov tests by 2 52

Figure 22 – Declines by replicating the FailOther tests by 2 53

Figure 23 – Pseudo code of the findLocalMaxima algorithm 56

Figure 24 – Application of Local Maxima for Jaccard on PrintTokens_v01 with Distinct Test Suite 58

Figure 25 – Application of Local Maxima for Tarantula on PrintTokens_v01 with Distinct Test Suite 59

Figure 26 – Application of Local Maxima for Ochiai on PrintTokens_v01 with Distinct Test Suite 60

Figure 27 – Application of Local Maxima for Ela on PrintTokens_v01 with Distinct Test Suite 61

Figure 28 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Redundant Test Suite 65

Figure 29 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Redundant Test Suite 66

Figure 30 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct Test Suite 69

Figure 31 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct Test Suite 70

Figure 32 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Test Suite 73

Figure 33 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Test Suite 74

Figure 34 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Cluster Test Suite 77

Figure 35 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Cluster Test Suite 78

Figure 36 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Classify Test Suite 81

Figure 37 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Classify Test Suite 82

Figure 38 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Redundant Test Suite 85

Figure 39 – LM Improvement over Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques for seven programs on Redundant Test Suite 87

Figure 40 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct Test Suite 88

Figure 41 – LM Improvement over Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques for seven programs on Distinct Test Suite 90

Figure 42 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Test Suite 91

Figure 43 – LM Improvement over Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques for seven programs on Distinct FminCov Test Suite 92

Figure 44 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Cluster Test Suite 94

Figure 45 – LM Improvement over Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques for seven programs on Distinct FminCov Cluster Test Suite 95

Figure 46 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Classify Test Suite 97

Figure 47 – LM Improvement over Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques for seven programs on Distinct FminCov Classify Test Suite 98

LIST OF ABBREVIATIONS

ANSI	:	American National Standards Institute
BI	:	Bug Isolation
CC	:	Coincidentally Correct
CT	:	Cause Transition
CTB	:	Crosstab Based
DT	:	Decision Tree
FMINCOV	:	Fault with Minimum Coverage
GCC	:	GNU Compiler Collection
HMDS	:	Hierarchical Multidimensional Scaling
IR	:	Information Retrieval
KNN	:	K Nearest Neighbors
LM	:	Local Maxima
LOC	:	Line of Code
MDS	:	Multi Dimensional Scaling
MKBC	:	Minus and Key Block Chain
MRR	:	Mean Reciprocal Rank
NIST	:	National Institute of Standards and Technology
NN	:	Nearest Neighbor
SD	:	Set Difference
SFL	:	Statistical Fault Localization
SI	:	Set Intersection
SIR	:	Software-artifact Infrastructure Repository
SU	:	Set Union
SVM	:	Support Vector Machine

CHAPTER 1

INTRODUCTION

Software debugging is an expensive and time consuming task since it is generally a manual process (Newman, 2002; Wong & Debroy, 2009; Srivastav, Singh, & Chauhan, 2010). Software debugging involves the process of locating software faults, finding their causes, and fixing them. Among all these activities, the fault localization is the most difficult one. Fault localization is the activity of identifying the exact locations of program faults during program debugging (Wong & Debroy, 2009). To automate or semi automate this difficult activity with minimum human intervention, a smart fault localization technique is always needed. Unfortunately, the techniques that claim to effectively locate software faults have not yet reached at a desired level (Wong, Debroy, & Xu, 2012; Sahoo, Criswell, Geigle, & Adve, 2013).

Many techniques have been proposed for automating the fault localization process in the literature. These techniques can be categorized into traditional debugging techniques (Balzer, 1969; Agrawal, De Millo, & Spafford, 1991), algorithmic debugging techniques (Shapiro, 1983), program slice based techniques (Agrawal, 1991; Agrawal, Horgan, London, & Wong, 1995; Renieris & Reiss, 2003), memory state based techniques (Cleve & Zeller, 2005), statistical techniques (Jones & Harrold, 2005; Liu, Yan, Fei, Han, & Midkiff, 2005; Abreu, Zoeteweij, & Van Gemund, 2006; Jeffrey, Gupta, & Gupta, 2008; Parsa, Vahidi-Asl, Arabi, & Minaei-Bidgoli, 2009; Wong et al., 2012; Xu, Chan, Zhang, Tse, & Li, 2011; Zhang, Chan, Tse, Yu, & Hu, 2011), and slice-based statistical techniques (Mao, Lei, Dai, Qi, & Wang, 2014; Ju et al., 2014).

Among these techniques, the statistical fault localization techniques (SFL) are the most commonly used ones since they are lightweight and do not require users to provide additional information such as the structure of the program. Statistical techniques analyze the

relationship between the fail/pass results of tests and the program elements (e.g. statements) executed by these tests. Based on the test results and corresponding execution information of program elements, each SFL uses a different way to assign a suspiciousness value for the program elements. There are some representative techniques such as Tarantula (Jones, Harrold, & Stasko, 2002), Jaccard (Abreu et al., 2006), and Ochiai (Abreu et al., 2007) which are the most well known the statistical fault localization techniques (Kim & Lee, 2014). The effectiveness of these techniques is still a bottleneck due to the abstraction level. Although Jaccard, Tarantula, and Ochiai include the statement coverage of passed tests to their calculations of suspiciousness values, they assign same weights to the statement coverage information of both kinds of tests. There can be some cases in which the statement coverage information of failed tests are the same but the statement coverage information of passed tests are different. In these cases, the existing techniques do not differentiate the faulty statement from the innocent statements. The effectiveness of fault localization could be improved including such information during the process.

Together with the fault localization effort, the size of the test suite affects the time spent during software debugging process. Although test suite reduction is mostly studied for regression testing, there are several recent studies that evaluate the effect of test suite reduction on fault localization techniques. However, there is no consensus on the effects of test suite reduction on fault localization. A number of studies proposing test suite reduction strategies show that the test reduction results in a decrease in the fault localization effectiveness (Yu, Jones, & Harrold, 2008; Rothermel, Harrold, Ostrin, & Hong, 1998; Hao et al., 2010; Baudry, Fleurey, & Le Traon, 2006). There are also other studies showing that the test reduction results in an improvement in the fault localization effectiveness (Masri & Assi, 2014; Dandan, Tiantian, Xiaohong, & Peijun, 2013; Hao, Zhang, Zhong, Mei, & Sun, 2005a). They suggest excluding the passed tests that may obscure the process based on several criteria. Most of the techniques in the literature apply clustering, classification, and sampling of failed and passed tests without considering whether they are good or bad tests, i.e. without considering whether they contain valuable statement coverage information or not.

We propose a fault localization framework that introduces a new fault localization metric called Ela, three test suite reduction strategies to improve the effectiveness of fault localization, and an effective ranking strategy to improve the ranking of statements. We focus on the situations not covered by the existing fault localization techniques and test suite

reduction techniques to improve the fault localization effectiveness and decrease the fault localization effort by this framework.

We propose a new metric for statistical fault localization. The intuition is that a faulty statement is more frequently executed by the failed tests and less frequently executed by the passed tests. We also differ from the existing techniques in expressing these frequencies and their combinations. We performed several experiments to evaluate the proposed metric on the Siemens suite (Hutchins, Foster, Goradia, & Ostrand, 1994) available at the software-artifact infrastructure repository (SIR, 2014). The Siemens suite contains seven programs: PrintTokens, PrintTokens2, Replace, Schedule, Schedule2, Tcas, and TotInfo. Due to its high quality, many researchers studying the fault localization have performed their experiments on this test suite (Renieris & Reiss, 2003; Cleve & Zeller, 2005; Jones & Harrold, 2005; Liu et al., 2005; Abreu et al., 2006; Jeffrey et al., 2008; Parsa et al., 2009; Wong et al., 2012; Zhang et al., 2011). In our experiments, we have used the three popular fault localization techniques, which are Jaccard, Tarantula, and Ochiai coefficients, as a base line. The experimental results show that the proposed technique outperforms these three prominent techniques. During the comparison, in addition to using the widely accepted expense and code examination effort metrics, we adapt a metric of the information retrieval domain, called Mean Reciprocal Rank (MRR), to assess the overall ranking quality of the SFL techniques. The proposed technique has a higher ranking than others for 3.4% (4 of 118) of all the versions of the subject programs while it is one of the best performing techniques for 96.6% (114 of 118) of all the versions of the subject programs.

We apply equivalent test elimination strategy (Distinct Test Suite Strategy) to achieve more accurate fault localization whereas most of the statistical fault localization techniques do not consider the bias caused by the existence of the equivalent tests. If two tests execute the same set of statements, then we consider them as equivalent tests with respect to their statement coverage. We assume that there are several bad tests in the test suite: bad passed tests and bad fail tests. A bad passed test is defined as “the test that passes even if it executes the faulty statement”. A bad fail test is defined as “the test that executes too many innocent statements”. We empirically show that increasing the number of tests that are equivalent to a bad test has a negative effect on the result of the fault localization. The details are given in the subsection 5.4. Since it is not known whether a test is a bad test or not, the existence of the equivalent tests can cause a bias in the ranking of statements according to their suspiciousness values during the process of statistical fault localization. Therefore, we select the safe side and eliminate the equivalent tests from both passed and failed tests in our

experiments. This strategy achieves on average 99.5% test size reduction. Moreover, the proposed technique has a higher ranking than others for 26.3% (31 of 118) of all the versions of the subject programs while it is one of the best performing techniques for 73.7% (87 of 118) of all the versions of the subject programs with this Distinct Test Suite Strategy.

We propose a new test suite reduction strategy to reduce the effort for the fault localization by reducing the test suite size. Different from the literature, we propose to eliminate the failed tests that may mislead the SFL to assign higher suspiciousness values to innocent program elements (FminCov Test Suite Strategy). We empirically show the effects of this strategy on three popular SFL techniques (Jaccard, Tarantula, and Ochiai) by using the Siemens suite. This strategy achieves on average 10.1% test size reduction.

In addition, we investigate two kinds of reductions of the passed tests in combination with the FminCov Test Suite Strategy. In the first kind of reduction, we aim to eliminate the passed tests that are close to the eliminated failed tests, similar to Masri and Assi (2014). The intuition is to remove the tests that execute the faulty statement but still pass (Classification Strategy). We first classify the passed tests into the eliminated failed tests and the remaining failed tests by using KNN classification algorithm. Then select the class that contains the remaining failed tests and remove other class. This strategy achieves on average 30.3% test size reduction. In the second kind of reduction, similar to Dandan, Tiantian, Xiaohong, and Peijun (2013), we aim to eliminate the passed tests that are not very close to the remaining failed tests (Clustering Strategy). The intuition is that of delta debugging (Zeller, 1999) which states that a passing run closest to a failing run contains the most information. We cluster the remaining failed tests and all the passed tests into subsets by using a hierarchical clustering algorithm, specifically Agglomerative clustering, and then select the subset that contains the remaining failed tests. This strategy achieves on average 34.1% test size reduction.

In our experiments, we examine all three test suite reduction strategies and show that all the reduction strategies result in significant reductions in the size of tests. Among all three, the failed test elimination strategy results in the best improvement but the elimination of the passed tests similar to the eliminated failed tests (Classification Strategy) results in quite comparable results to failed test elimination strategy. Removing the passed tests similar to the eliminated failed tests results in high reductions in the size of test suite (up to 81%). Best of these test suite reduction strategies achieves an improvement up to 1.7% in Jaccard, up to

2.46% in Tarantula, up to 1.01% in Ochiai, and up to 0.38% in Ela in terms of average expense.

We propose a new effective ranking strategy for improving the ranking of statements (Local Maxima Strategy) in order to improve the effectiveness and decrease the effort for the fault localization. Instead of serving all the statements with their suspiciousness ranks to the software developers, we aim to serve only the statements which are the local maximum in its 1-nearest neighborhood. This strategy reduces the number of statements that software developers must inspect to locate the fault. It assumes that the innocent statements near to the faulty statement are likely to be assigned with high suspiciousness values and should be eliminated from the list of suspicious statements. This strategy achieves an improvement 10.54% in Jaccard, 10.47% in Tarantula, 10.74% in Ochiai, and 10.88% in Ela in terms of average expense.

1.1. Research Questions

This dissertation proposes a fault localization framework that consists of a new fault localization metric called Ela, three test suite reduction strategies to improve the effectiveness of fault localization, and an effective ranking strategy to improve the ranking of statements. To achieve these goals four main research questions are answered.

RQ.1: Can we define a metric to achieve better fault localization accuracy?
RQ.2: Does the redundancy of tests affect the fault localization accuracy? What is the effect of equivalent tests?
RQ.3: Can we improve test suite quality?
- RQ.3.1: What type of tests affect test suite quality?
- RQ.3.2: Can we find a subset of tests that have better test suite quality by clustering and classification of failed and passed tests?

RQ.4: How can we improve the ranking of statements? Can we implement a post processing technique to improve the fault localization accuracy based on suspiciousness values of statements?

1.2. Contributions of the Dissertation

The contributions of this dissertation are as follows:

- A new fault localization metric: Ela.
- Empirical evidence showing that equivalent tests distort the result of fault localization and a distinct test selection strategy.
- Three test suite reduction strategies and their empirical evaluations.
- A new effective suspiciousness ranking strategy: Local Maxima.

1.3. Structure of the Dissertation

The rest of this dissertation is organized as follows: Section 2 gives an overview of the fault localization techniques in the literature. Section 3 introduces the preliminaries of the fault localization: the suspiciousness metrics (Jaccard, Tarantula, and Ochiai), the evaluation metrics: expense, code examination effort, and mean reciprocal rank. Section 4 describes the proposed fault localization technique: Ela. Section 5 explains the proposed three test suite reduction strategies. Section 6 explains the proposed three test suite reduction strategies. Section 7 presents the experiments, the comparisons and the results. Section 8 discusses the threats to the validity of the research. Section 9 presents the conclusions and the discussions. Section 10 discusses the directions for the future work.

CHAPTER 2

LITERATURE REVIEW

Fault localization is a research topic which aims to reduce the effort and the cost of the software projects by improving the rate of fault detection. Many techniques have been proposed for automating the fault localization process in order to achieve this aim. In this chapter, we first review the fault localization techniques, and then present the current state-of-the-art for test suite reduction and test suite selection techniques applied on the fault localization in the literature.

2.1. Fault Localization

In this study, we categorize the fault localization techniques into five categories: traditional debugging technique, algorithmic debugging technique, program slice based technique, memory state based technique, and statistical technique. The studies proposed in the literature under these categories are covered in the following subsections.

2.1.1. Traditional Debugging Technique

Software developers traditionally use two different approaches for finding the software faults. In the first one, software developers insert print statements into the subject programs and monitor whether the program executions reach to these print statements or not. In addition, they can output several variables and their values in the print statements to show the runtime behavior of the subject programs. In the second one, software developers use the symbolic debuggers to debug the subject programs. They can stop at a particular point and examine the current states of the variables in the programs. In addition, they can change the current values of the variables and follow the runtime behavior of the programs by using these symbolic debuggers (Balzer, 1969; Agrawal et al., 1991).

2.1.2. Algorithmic Debugging Technique

In this technique, complex computations are decomposed into the simpler sub-computations. Software developers check each of the sub-computations for correctness. When a complex computation is incorrect then its sub-computations are checked for correctness. If a sub-computation is determined as incorrect, then it is assumed that the fault is localized in this sub-computation. On the other hand, a complex computation can be incorrect while all of its sub-computations are correct. In this case, it is assumed that the fault is located in the compositions of the sub-computations (Shapiro, 1983).

2.1.3. Program Slice Based Technique

It is a code based technique used for software debugging and fault localization. Execution slice is defined as the set of statements executed by a program for a particular test (Agrawal, 1991). In this technique, the initial set of suspicious statements is generated by removing the statements executed by the passed tests from the set of statements executed by the failed tests. The techniques set difference (SD), set union (SU), set intersection (SI), and nearest neighbor (NN) are four different specific program slicing techniques.

2.1.3.1. Set Difference Technique (SD)

In this technique, one passed and one failed tests are used. The set of the statements executed by a passed test is removed from the set of the statements executed by a failed test. The resulting set is used as the initial set of suspicious statements for finding the software faults (Agrawal et al., 1995).

$$E_{initial} = E_{failed} - E_{passed} \quad (1)$$

Agrawal, Horgan, London, and Wong (1995) present a fault localization tool called χSlice (χSuds, 1998) based on slicing and dicing for the standard C programming language. An experiment is performed on a complex UNIX sort program to show the usefulness of slicing in locating software faults. They seed the sort program one at a time with a total of 25 bugs resulting in 25 erroneous variants of the sort program with a single bug. The experimental results show that χSlice is an effective tool in locating software faults since the ratio of the block coverage and the decision coverage are 96% and 89% respectively.

2.1.3.2. Set Union (SU) & Set Intersection (SI) Techniques

The set union and the set intersect techniques are two specific approaches of the program slicing techniques (Renieris & Reiss, 2003). The union of the statements executed by all passed tests is removed from the set of the statements executed by a single failed test.

$$E_{initial} = E_{failed} - \cup E_{passed} \quad (2)$$

The set of the statements executed by a single failed test is removed from the intersection of the statements executed by every passed test.

$$E_{initial} = \cap E_{passed} - E_{failed} \quad (3)$$

The resulting set is used as the initial set of suspicious statements for finding the software faults (Renieris & Reiss, 2003).

2.1.3.3. Nearest Neighbor (NN) Technique

In this technique, any single failed test is selected first and then the passed test having most similar coverage to the selected failed test is found. Afterwards, the set of the statements executed by the found passed test is removed from the set of the statements executed by the selected failed test. The resulting set is used as the initial set of suspicious statements for finding the software faults (Renieris & Reiss, 2003).

$$E_{initial} = E_{failed} - E_{passed} \quad (4)$$

The authors perform an experiment on the Siemens test suite. The NN technique is compared with SU and SI techniques on this test suite. The experimental results show that the NN technique outperforms SU and SI techniques.

2.1.4. Memory State Based Technique

Cleve and Zeller (2005) propose a cause transition (CT) technique to locate the failure inducing faults. This technique performs a binary search of the memory states of a program between a passed test and a failed test. It defines a method to automate the process of making hypotheses about how state changes affect the output of the program. The authors compare their technique with the NN technique on the Siemens suite. The experimental results show that their technique stops searching with the 10% code examination effort for 35.66% of all the faulty versions of the subject programs while the NN technique stops searching with the 10% code examination effort for 16.51% of all the faulty versions of the subject programs. Therefore, their technique outperforms the NN technique.

2.1.5. Statistical Technique

The coverage information on different entities such as classes, methods, blocks, branches, predicates, statements etc. are used in the statistical fault localization techniques. First, the suspiciousness values for these entities based on their coverage information are calculated.

Then, these entities are ranked according to their suspiciousness values. Afterwards, software developers investigate these entities according to their ranks in order to find the software faults. The studies using the statistical fault localization techniques are categorically introduced in the following subsections.

2.1.5.1. Predicate Based Technique

Liu, Yan, Fei, Han, and Midkiff (2005) propose SOBER which uses the predicate coverage information to localize the software faults without any prior knowledge of the program semantics. The SOBER approach models the evaluation patterns of predicates in both correct and incorrect runs respectively. An experiment is conducted on the programs of the Siemens test suite. The accuracies of the SOBER and two statistical fault localization techniques CT and BI are compared on this test suite. The experimental results show that the SOBER approach can help software developers locate the software fault for the 68 out of 130 faulty versions of the subject programs while the better of two statistical fault localization techniques can help software developers locate the software faults for the 52 out of 130 faulty versions of the subject programs with less than or equal to the 10% code examination effort. As a result, their approach outperforms these two techniques in terms of code examination effort which is the percentage of faults located.

Zhang, Chan, Tse, Yu, and Hu (2011) offer a predicate based fault localization framework based on the three types of hypothesis testing methods. The first method includes non-parametric tests (Mann-Whiney test and Wilcoxon Signed Rank test), the second method includes standard parametric tests (F-test and t-test), and the third method includes debugging specific parametric tests (BI and SOBER). An experiment is carried out on the Siemens test suite to compare the three methods with each other. Mann-Whitney and Wilcoxon Signed Rank tests reach the most relevant predicate for 5.41% and for 17.2% of all the faulty versions of the subject programs respectively with the 10% code examination effort. F-test and t-test reach the most relevant predicate for 1.8% and for 4.5% of all the faulty versions of the subject programs respectively with the 10% code examination effort. BI and SOBER techniques reach the most relevant predicate for 9.01% and for 8.11% of all faulty versions respectively with the 10% code examination effort. Therefore, non-parametric testing method outperforms the other two methods.

Parsa, Vahidi-Asl, Arabi, and Minaei-Bidgoli (2009) propose a statistical debugging approach based on elastic net. This approach first finds the smallest effective subset of program predicates known as bug predictors and then detects the main causes of the faults by

using backward slicing technique. When the number of executions is much smaller than the number of predicates, this approach is more advantageous since it reduces the set of program predicates into the smallest effective subset of program predicates. A linear regression model is constructed to find the relationship between program predicates and the falling and passing state of the program. An experiment is performed on the Siemens test suite and a real life image processing program EXIF. Each program in the test suite is instrumented by a small code to collect the values of the program predicates. Their approach is compared with the statistical fault localization techniques SOBER and BI. The experimental results show that the elastic net approach finds 92 out of 132 faulty versions of the subject programs while SOBER and BI find 68 and 53 out of 132 faulty versions of the subject programs respectively with less than or equal to the 10% code examination effort. As a result, their approach outperforms two statistical fault localization techniques SOBER and BI on the subject programs.

2.1.5.2. Statement Based Technique

Jones and Harrold (2005) present the Tarantula (Jones et al., 2002) fault localization technique which uses the statement coverage and compares it with the fault localization techniques from the literature such as SI, SU, NN and CT. Their experimental results show that the Tarantula technique is able to guide software developers to the faults for 13.93% of all the faulty versions of the subject programs while the best of other techniques is able to guide software developers to the faults for 5.43% of all the faulty versions of the subject programs with less than or equal to the 10% code examination effort on the Siemens test suite. Therefore, their technique is about twice more effective than the best of other techniques for finding faulty statements.

Wong, Debroy, and Xu (2012) present a crosstab based (CTB) statistical technique which uses the executions of executable statements for each test and the result of each test (success or fail) in order to localize software faults in an efficient and effective manner. Several experiments are performed on small size programs in the Siemens test suite and the UNIX test suite and on large size programs such as Space, Grep, Gzip, and Make. Each faulty version of the subject programs is instrumented by the instrumentation tool χSuds (χSuds, 1998) to collect their statement coverage information. Crosstab is constructed for each statement in order to determine its suspiciousness ratio. $\chi 2$ test is used to determine the associations between the statement coverage information and the result of tests. This technique is compared with the Tarantula, SOBER and BI. The experimental results show

that the CTB statistical technique outperforms SOBER and BI techniques since it is more efficient in time spent for locating the faults.

Jeffrey, Gupta, and Gupta (2008) present a value profile replacement approach for ranking the program statements according to their likelihood of being faulty. They propose a new value profile based approach for fault localization which involves searching for the program statements that can affect the output of a failing run such that an incorrect output becomes a correct output. This is done by replacing the values used in a statement during the execution of a failing run with an alternate set of values and checking whether the resulting output becomes a correct output or not. They compare their technique with the Tarantula on the Siemens test suite. The experimental results show that the value profile replacement approach locates software faults for 10.85% of all the faulty versions of the subject programs while the Tarantula approach locates software faults for 2.33% of all the faulty versions of the subject programs with less than or equal to the 10% code examination effort. As a result, their approach outperforms the Tarantula approach.

2.1.5.3. Method Based Technique

Dallmeier, Lindig, and Zeller (2005) propose a plug-in called Ample that helps software developers to locate the causes of failure in the Java programs. Ample works by comparing the method call sequences of the passing tests with the sequences of the failing tests. A difference in the method call sequences is assumed to be likely to locate the erroneous class. It presents the ranking of the classes which are likely to be responsible for the failure. Therefore, software developers looking for the bugs are advised to inspect the classes in the presented order. They perform an experiment on the NanoXml parser to evaluate their rankings. Their experimental results show that the defective class is immediately identified in 36% of all test runs. It is stated that a software developer using their technique must inspect on average 21% of the executed classes (10% of all classes) before finding the bug.

2.1.5.4. Block Based Technique

Abreu, Zoeteweij, and Van Gemund (2006) propose a block coverage based fault localization technique. It is aimed to localize the software faults by using different similarity coefficients. The coefficients are selected from the automated fault localization techniques Jaccard, Tarantula, Ample (Dallmeier, Lindig, & Zeller, 2005), and Ochiai (Abreu et al., 2007). An experiment is performed on the Siemens test suite. Every single program in the test suite is instrumented by the instrumentation tool called Front (Augusteijn, 2002). The effectiveness of selected coefficients in terms of code examination effort is evaluated on this

test suite. The experimental results show that the Ochiai coefficient decreases the percentage of the code blocks needed to be inspected by 5% and outperforms the other three coefficients.

Xu, Chan, Zhang, Tse, and Li (2011) present the Minus and Key Block Chain (MKBC) fault localization technique which is based on the chains of key basic blocks. The MKBC technique is compared with five fault localization techniques, which are Minus (Xu et al., 2011), Jaccard, Ochiai, BI, and Tarantula, on three real life medium size programs (Jtopas, Xml-security, and Ant) from the software-artifact infrastructure repository (SIR, 2014). Each program in the test suite is instrumented by the instrumentation tool called Soot (Vallée-Rai et al., 1999). The experimental results show that the MKBC localize software faults for 10.35% of all the faulty versions of the subject programs while Minus, Jaccard, Ochiai, BI, and Tarantula techniques localize software faults for 3.45%, 3.45%, 3.45%, 0%, and 3.45% of all the faulty versions of the subject programs respectively with less than or equal to 1% code examination effort. Therefore, the MKBC technique outperforms other five techniques in terms of the code examination effort.

2.2. Test Suite Reduction and Test Suite Selection

Test suite reduction and test selection have gained a great attention in recent years in fault localization community. There are several studies proposing new test suite reduction strategies and evaluating their effects on fault localization in the literature. Currently, there is no consensus on the effects of test suite reduction and test suite selection on the fault localization. The studies have showed that these strategies resulted in a decrease of fault localization effort and achieved an improvement over fault localization effectiveness.

Yu, Jones, and Harrold (2008) investigate the effect of six reduction strategies, which are variations of removing the tests that execute the same statements (coverage vector-based reduction), and variations of removing the tests that do not contribute to the statement coverage (statement-based reduction). They show that the statement coverage based reduction causes extensive reduction in the effectiveness whereas the coverage-vector based elimination improves the effectiveness. Hao et al. (2010) propose test reduction strategies that select a subset of a test suite with minimal undistinguishable statements. They assume that two statements are undistinguishable if every test executes both of them or neither of them. In one of their strategies, they extend the definition of undistinguishable statements to the statements which are executed by the same number of failed and passed tests. Their

experimental results show that these reductions cause minor decrease in fault localization effectiveness.

Abreu, Zoeteweij, and Van Gemund (2007) and Baudry, Fleurey, and Le Traon (2006) investigate the effect of the number of tests in fault localization. Baudry et al. (2006) enhance the existing test suite by adding new tests that increase dynamic basic blocks which are set of statements covered by the same tests. They state that adding such tests increase the fault localization effectiveness. Abreu et al. (2007) select subsets with varying numbers of passed and failed tests. Their experimental results show that the effect of adding more than six failed tests or more than twenty passed tests results in minimal impact on the fault localization effectiveness.

There are also several studies showing that test reduction improves fault localization effectiveness. Dandan et al. (2013) propose a two-step reduction. They first remove the passed tests whose coverage vector is orthogonal to that of all failed tests for programs with a single fault. In multi fault localization, the passed tests which are orthogonal to at least one failed test are removed. Then, they select representatives with minimal execution path length for each group of tests with identical coverage vectors. Hao et al. (2005b) claims that redundant tests in the test suite may cause a bias and harm the SFL effectiveness. They propose similarity aware fault localization based on the application of fuzzy sets. In their experiments reported in Hao, Zhang, Zhong, Mei, and Sun (2005a), the elimination of redundant tests improves the fault localization effectiveness. They use the technique proposed by Harrold, Gupta, and Soffa (1993) but do not state whether they use the statement coverage as test requirement or def-use pairs as in the original paper. Masri and Assi (2014) define the term coincidental correctness for a test when it executes the faulty statement but still produces the correct result (i.e. passed test). They propose several techniques to remove such tests by using a clustering based approach where they eliminate the passed tests similar to failed ones. They achieve better fault localization with such elimination for some cases. However, they remove some of the most informative tests with this approach as well. Zeller (1999) states that a passed test similar to a failed test contains most information.

Podgurski, Masri, McCleese, Wolff, and Yang (1999) present a stratified random sampling approach to reduce the number of program executions that software developers should check manually in the testing process. They first collect the profiles of the programs executions. Next, they cluster these program executions according to the similarities among their

profiles. Afterwards, they perform a stratified random sampling to estimate the proportion of failures in the entire execution population for the program. In this sampling, a random sample is selected from each cluster without replacement. They perform their experiments on five ANSI C parsers and a project scheduling system. They compare the efficiency of the stratified random sampling with simple random sampling. The execution population for each of five parsers is clustered into approximately 100, 150, 200, and 250 strata (5x4=20 cases). Moreover the execution population for the project scheduling system is clustered into approximately 100, 150, and 200 strata (6x3=18 cases). They use a two stage clustering approach which combines partitioning and hierarchical clustering algorithms. In the first stage, they partition the executions into first-stage clusters by using k-medoids algorithm. In the second stage, they cluster the first-stage clusters into second-stage clusters by using hierarchical clustering algorithm. Finally, they select a random sample from each second-stage cluster without replacement. Experimental results show that the stratified random sampling is more accurate than simple random sampling for 97.36% of all cases (37 of 38).

Dickinson, Leon, and Podgurski (2001a) evaluate the effectiveness of the cluster analysis of execution profiles to find failures among the program executions. They compare several filtering procedures which involve a clustering strategy and a sampling strategy. They perform their experiments on five programs which are a word count program, a directory listing program, a regular expression parser, a regular expression finder, and a java pretty printer. They use agglomerative clustering algorithm and five sampling strategies which are random sampling, one-per-cluster sampling, adaptive sampling, n-per-cluster sampling, and small cluster sampling in their experiments. They use 1%, 2.5%, 5%, 10%, 15%, 20%, 25%, and 30% percentages of the size of the execution population as the cluster counts. Experimental results indicate that adapting sampling strategy is more efficient for finding failure than other four sampling strategies for all program variations.

Dickinson, Leon, and Podgurski (2001b) propose a failure pursuit sampling which is a form of adaptive sampling for revealing failures in the software. They define a cluster filtering procedure which involves selecting the clustering algorithm, the dissimilarity metric, the cluster count, and the sampling method. They perform their experiments on GNU LilyPond music typesetting program (Nienhuys & Nieuwenhuizen, 2003) and the C-language compiler of the GNU Compiler Collection (GCC) (Stallman, 2009) to evaluate the failure pursuit sampling. The executions are partitioned into the clusters by using the agglomerative hierarchical clustering algorithm. N-dimensional Euclidean distance is used as the dissimilarity metric. 1%, 2.5%, 5%, 10%, 15%, 20%, 25%, and 30% percentages of the size

of the execution population are used as the cluster counts. Five different sampling methods which are simple random sampling, one-per cluster sampling, n-per-cluster sampling, small cluster sampling, and adaptive sampling are used to select the executions from clusters. Experimental results show that the adaptive sampling finds more failures than the simple sampling, the one-per-cluster sampling, and n-per-cluster sampling. Moreover, failure-pursuit sampling and adaptive sampling find similar numbers of failures in the experiments. Therefore, they conclude that the failure-pursuit sampling is as effective as adaptive sampling.

Podgurski et al. (2003) propose automated support for classifying failed executions in order to prioritize them and diagnose their causes. First, they combine the failed executions with a random sample of successful executions. Next, they apply the feature selection strategy to select the features used to distinguish failed executions from successful executions. These features are actually pattern classifiers. Logistic regression models are employed as pattern classifiers in this selection strategy. Afterwards, they apply the classification strategy to group failed executions whose execution profiles are similar according to the selected features. They use k-medoids clustering algorithm and hierarchical multidimensional scaling (HMDS) multivariate visualization technique. The resulting classification of failed executions is used to find the faults manually in the subject programs. They perform their experiments on three large subject programs which are GCC (Stallman, 2009), Jikes (Jikes, 2014), and Javac (Java, 2014) to evaluate their classification strategy and compare it with manual classification. Experimental results indicate that their classification strategy is effective and scales to large programs.

Masri and Assi (2014) propose an approach that groups the passing tests into two clusters which are true passing tests and coincidentally correct tests. There are actually three clusters which are true passing tests, failing tests, and coincidentally correct tests in their approach. A coincidentally correct (CC) test is defined as the test in whose executions the program produces coincidentally the correct output. In other words, the fault is executed but its execution does not take effect on the result of test. They present two kinds of techniques which are tech-I and tech-II to find the CC tests. In tech-I, they partition all of passed and failed tests into two clusters by using k-means clustering algorithms. Hereafter, they select the cluster which contains the majority of the failing tests and label the passing tests within this cluster as CC tests. In tech-II, they partition only the passing tests into two clusters by using k-means clustering algorithms. Hereafter, they select the cluster which has higher relevance and label the passing tests within this cluster as CC tests. They perform their

experiments on several subject programs such as PrintTokens, PrintTokens2, Replace, Schedule, Schedule2, Tcas, TotInfo, NanoXml parser v.1, NanoXml parser v.3, NanoXml parser v.5, Space, Sed, Flex, and Gzip from SIR (SIR, 2014) and JTidy (JTidy, 2014). Their experimental results show that their technique is promising.

Farjo and Masri (2014) propose three heuristics which assign weights to the profiling elements such that higher weights indicate more potential relevance to failure. Heuristic-I assigns higher weight to the profiling elements not covered by many tests, since the number of failing tests is typically much smaller than the number of passing tests. Heuristic-II extends Heuristic-I by using a fuzzy logic approach in the computation of the weights. Heuristic-III assigns higher weights to the profiling elements covered by tests which are very dissimilar from others since failing tests are likely to be outliers. They perform several experiments on several subject programs such as PrintTokens2, Schedule, TotInfo, Space, Flex, Sed, and Gzip from SIR (SIR, 2014), Tomcat (Tomcat, 2014), Jigsaw (Jigsaw, 2014), and JTidy (JTidy, 2014). They empirically evaluate their heuristics by measuring their impact on an established test suite minimization technique. Experiment results indicate that the third one exhibits the most positive impact on the execution profiles. Although their results are not positive in all cases, they are encouraging to investigate more heuristics.

2.3. Surveys about Fault Localization, Test Suite Reduction and Test Suite Selection

There are several survey papers about fault localization, test suite reduction, and test suite selection in the literature. In this section, we provide a brief summary of these papers.

Wong & Debroy (2009) provide an overview of various fault localization methods in the literature. They categorize them in seven categories: “static, dynamic and execution slice-based”, “program spectrum-based”, “statistics-based”, “program state-based”, “machine learning-based”, “model-based”, and “data mining-based”. Furthermore, they discuss an effectiveness metric called “Exam” for fault localization.

Alipour (2012) categorizes the fault localization techniques based on their common features into five categories: “program slicing”, “spectra based fault localization”, “statistical inference”, “delta debugging”, and “model checking”. He examine several important techniques for automated fault localization under “delta debugging”, and “model checking” categories in the literature. Moreover, he briefly discuss the merits and shortcomings of these techniques.

Agarwal & Agarwal (2014) review various articles, journals and conference papers on software fault localization in the literature. They aim to give essential information about the methods, dataset, and techniques which are used for comparison. They state that coverage based methods gain significant attention and should be used to a large extent. They specify that large datasets achieve more accurate results and should be used to evaluate the methods.

Su, Gong, Wang, and Ma (2014) discuss three approaches: test case reduction, fault localization, and fault comprehension. They analyze current representative techniques about three approaches and their limitations. They also discuss on-going research issues about three approaches.

Abreu et al. (2006) studies the influence of four similarity coefficients: Jaccard, Tarantula, Ample, and Ochiai. They evaluate four coefficients and assess their effectiveness by using block coverage information on the Siemens suite. They conclude that Ochiai coefficient consistently outperforms Jaccard, Tarantula, and Ample coefficients.

Kim and Lee (2014) identify the characteristics of the existing studies through the experimental analysis. Based on their characteristics, the existing studies are divided into three groups: Jaccard group, Ample group, and Ochiai group. They provide the strength and the weakness of each group.

Vidács, Beszédes, Tengeri, Siket, and Gyimóthy (2014) investigate the effect of different test reduction methods on the performance of fault localization and detection techniques. They also provide new combined methods that incorporate both localization and detection aspects. They empirically evaluate the methods by measuring detection and localization metrics of test suites with various reduction sizes, and by how reduced test suites perform with actual faults. They perform their experiments with SIR programs traditionally used in fault localization research, and extend the case study with large industrial software systems.

CHAPTER 3

PRELIMINARIES

Among the fault localization techniques, statistical fault localization technique is the most commonly used technique since it is lightweight and does not require its users to provide additional information such as the structure of the program. Statistical techniques analyze the relationship between the result of failed or passed tests and the statement coverage of these tests. There are some representative techniques such as Tarantula, Jaccard, and Ochiai based on the statistical fault localization techniques (Kim & Lee, 2014). These techniques are widely used in most of the studies in the literature. Therefore, these leading techniques can be compared to evaluate the accuracy of a new statistical fault localization technique. We call them the three prominent fault localization techniques in this study. Some example studies that use Jaccard, Tarantula, and Ochiai are given below.

- **Jaccard**: Abreu et al., 2006; Yu et al., 2008; Zhang, Chan, Tse, Jiang, & Wang, 2009; Xie, Wong, Chen, & Xu, 2010; Naish, Lee, & Ramamohanarao, 2011; Xu et al., 2011; Zhang et al., 2011; Artzi, Dolby, Tip, & Pistoia, 2012; Chan & Cai, 2012; Qi, Mao, Lei, & Wang, 2013; Xu, Zhang, Chan, Tse, & Li, 2013; Mao et al., 2014; Kim & Lee, 2014

- **Tarantula:** Jones & Harrold, 2005; Abreu et al., 2006; Liu, Fei, Yan, Han, & Midkiff, 2006; Jeffrey et al., 2008; Yu et al., 2008; Zhang et al., 2009; Hao et al., 2010; Xie et al., 2010; Baah, Podgurski, & Harrold, 2011; Naish et al., 2011; Wong et al., 2012; Zhang et al., 2011; Xu et al., 2011; Artzi et al., 2012; Chan & Cai, 2012; Maheswari & Venkatesakumar, 2013; Qi et al., 2013; Sahoo et al., 2013; Xu et al., 2013; Yoo, Harman, & Clark, 2013; Mao et al., 2014; Kim & Lee, 2014

- **Ochiai:** Abreu et al., 2006; Yu et al., 2008; Xie et al., 2010; Baah et al., 2011; Naish et al., 2011; Xu et al., 2011; Zhang et al., 2011; Artzi et al., 2012; Chan & Cai, 2012;

Maheswari & Venkatesakumar, 2013; Qi et al., 2013; Sahoo et al., 2013; Xu et al., 2013; Mao et al., 2014; Kim & Lee, 2014

Given a test suite and their pass/fail results, statistical fault localization techniques, a.k.a. spectrum-based or coverage-based fault localization, use the coverage information of failed and passed tests to determine the likelihood of a program element (e.g. classes, methods, blocks, branches, predicates, statements etc.) being faulty. This likelihood is called the *suspiciousness* of an element. The coverage type used in the spectra determines the unit of the faulty program element. For example, if the statement coverage is used, the technique determines the suspiciousness of the program statements. In this study, we are using the statement coverage as the execution information.

The coverage information of all tests are given to a fault localization technique is in the form of a coverage matrix as illustrated in Figure 1. Each row in this matrix is a coverage vector of a test in the test suite. Let *m* be the number of tests in the given test suite, *n* be the number of statements in the program, and let *i, j* be two integers where $0 < i \leq m$ and $0 < j \leq n$. The coverage matrix is a boolean $m \times n$ matrix where an entry A_{ij} shows whether the test *Ti* has executed the statement s_j or not. A_{ij} is 1 if the test run *Ti* has executed the statement s_j. The test results are represented with the result vector where each entry *Ri* shows whether test *Ti* is a passing run (1) or a failing run (0).

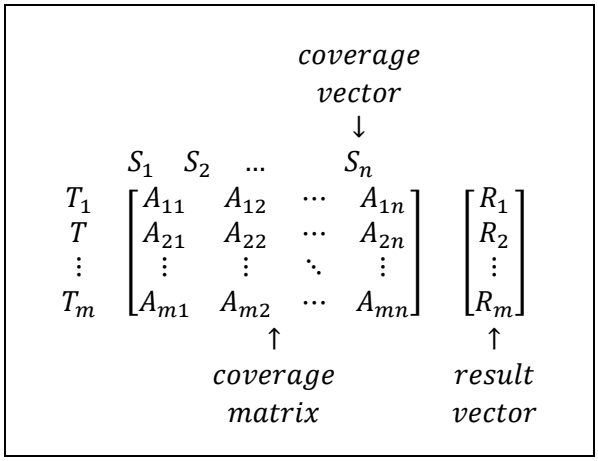

Figure 1 – Coverage matrix and result vector

m : Number of tests
n : Number of statements
S_n : Statement ‘n’
T_m : Test ‘m’
R_m : Result of the test ‘m’ (passed = 1; failed = 0)
A_{mn} : Coverage result of the statement ‘n’ by the test ‘m’ (executed = 1; not executed = 0)

Given such a coverage matrix and its corresponding result vector, each SFL technique determines the suspiciousness of program elements based on different intuitions. This suspiciousness is computed using the number of failed and passed tests that executed (or not executed) the statement. The basic notations used in the calculation of suspiciousness by three popular approaches which are Tarantula, Jaccard, and Ochiai coefficients are shown in Table 1.

Table 1 – Metrics used in the suspiciousness formulas of the fault localization techniques

A_f	Total # of failed tests
A_p	Total # of passed tests
$A_{ef}(s_j)$	Total # of failed tests executing statement 's_j'
$A_{nf}(s_j)$	Total # of failed tests not executing statement 's_j'
$A_{ep}(s_j)$	Total # of passed tests executing statement 's_j'
$A_{np}(s_j)$	Total # of passed tests not executing statement 's_j'

Using the suspiciousness values, the SFL technique ranks the program elements with respect to their likelihoods of containing the fault. There are five ranking strategies which are standard competition ranking (1-2-2-4 rule), modified competition ranking (1-3-3-4 rule), dense ranking (1-2-2-3 rule), ordinal ranking (1-2-3-4 rule), and fractional ranking (1-2.5-2.5-4 rule) (Lange, 2014). We use the standard competition ranking in this study.

3.1. Suspiciousness Metrics

In this section, we briefly introduce the three prominent fault localization techniques, Jaccard, Tarantula, and Ochiai. Three suspiciousness metrics used in the three prominent fault localization techniques are described as follows:

Jaccard

Abreu et al. (2006) used the Jaccard metric in their fault localization technique to localize the software faults. The Jaccard equation can be represented as:

$$Suspiciousness(s_j) = \frac{A_{ef}}{(A_{ef}+A_{nf}+A_{ep})} \quad (5)$$

Tarantula

Jones and Harrold (2005) used the Tarantula metric in their fault localization technique to localize the software faults. The Tarantula equation can be represented as:

$$Suspiciousness(s_j) = \frac{\frac{A_{ef}}{A_{ef}+A_{nf}}}{\frac{A_{ef}}{A_{ef}+A_{nf}} + \frac{A_{ep}}{A_{ep}+A_{np}}} \quad (6)$$

Ochiai

Abreu et al. (2007) used the Ochiai metric in their fault localization technique to localize the software faults. The Ochiai equation coefficient can be represented as:

$$Suspiciousness(s_j) = \frac{A_{ef}}{\sqrt{(A_{ef}+A_{ep}) x (A_{ef}+A_{nf})}} \quad (7)$$

3.2. Evaluation Metrics

Expense and Code Examination Effort are two metrics widely used to measure the effectiveness of the fault localization techniques in the literature (Renieres & Reiss, 2003; Cleve & Zeller, 2005; Jones & Harrold, 2005; Liu et al., 2005; Zhang et al., 2009).

3.2.1. Expense

Expense (Jones, 2008) measures the effort of a user to locate the faulty statement by inspecting the list of statements of a program in the order ranked by a fault localization technique. A fault localization technique presents a ranked list of statements as an output and the user is assumed to locate the faulty statement along this list. For example, if the rank of faulty statement is 'k' then the user inspects 'k' statements of the program to locate this faulty statement. The expense metric measures this cost. Expense can be represented as:

$$Expense = \left(\frac{\text{Rank of Fault}}{\text{Number of All Statements}}\right) x\ 100 \quad (8)$$

3.2.2. Code Examination Effort

Code Examination Effort (Jones & Harrold, 2005) is the number of faults located when examining a certain percentage of the source code.

$$Code\ Examination\ Effort = \left(\frac{\text{Percentage of Faults Located}}{\text{Percentage of Code Examined}}\right) \quad (9)$$

Suppose there are 4 versions and the expense for version 1 is 10%, version 2 is 20%, version 3 is 30%, and version 4 is 40%. Then the fault localization technique is said to catch 50% (2/4) of the faults in 20% of code examination effort. Similarly, the technique finds 75% (3/4) of the faults with 30% of code examination effort. Finally, the technique finds 100% (4/4) of the faults with 40% of code examination effort.

3.3. Mean Reciprocal Rank (MRR)

Mean reciprocal rank (Voorhees, 1999) is a measure used in the information retrieval (IR) domain for evaluating a process that returns a list of possible responses to a sample of queries, ranked by probability of correctness. In our case, we are evaluating a fault localization technique that returns a list of possible faulty statements, ranked by the probability of faultiness. To calculate the MRR, we first need to calculate the reciprocal rank which is defined as the multiplicative inverse of the rank of the first correct answer in IR domain. The reciprocal rank is the multiplicative inverse of the rank of the statement that actually has the fault in our case. Let 'P' be the set of subject programs, and FaultRank_p be the rank of the faulty statement reported by a technique for the subject program 'p' where $p \in P$. Then, the mean reciprocal rank for the technique is calculated as:

$$MRR = \frac{1}{|P|}\sum_{p \in P}^{N} \frac{1}{\text{FaultRank}_\text{p}} \tag{10}$$

3.4. Subject programs

In this study, we used the Siemens suite (Hutchins et al., 1994) available at the software-artifact infrastructure repository (SIR, 2014) as subject programs. We selected this suite since it is a quite frequently used benchmark suite in the fault localization and test reduction research (Jones et al., 2002; Renieres & Reiss, 2003; Cleve & Zeller, 2005; Abreu, Zoeteweij, & Van Gemund, 2007; Yu et al., 2008; Dandan et al., 2013). Siemens suite consists of seven C programs and associated test suites. For each of these programs, there are several versions, each of which contains manually injected one logical fault. There are 132 versions of C programs in this suite. For some of these versions, the test suite provided cannot differentiate the faulty version from the original program i.e. there were no failed tests for these versions. Since statistical fault localization techniques require at least one failed test, we excluded these versions in our experiments. We have used 118 versions of C programs as the subject programs in our study. Table 2 gives the following information about these seven C programs: program name, line of code, number of all tests, and a brief description of the program.

Table 2 – The seven C programs in SIR

Program	# of Versions	LOC	# of All Tests	Versions Excluded	Description
PrintTokens	7	565	4,130	V_{04} & V_{06}	lexical analyzer
PrintTokens2	10	529	4,115	No	lexical analyzer
Replace	32	563	5,501	No	pattern replacement
Schedule	9	412	2,650	V_{09}	priority scheduler
Schedule2	10	307	2,588	V_{04} & V_{09}	priority scheduler
Tcas	41	173	1,608	V_{13},V_{14}, V_{15},V_{36}, V_{38}	collision avoidance system
TotInfo	23	406	1,051	V_{06},V_{10}, V_{19},V_{21}	information measurer

The details about this suite are given in the subsection 7.1.

CHAPTER 4

PROPOSED TECHNIQUE: ELA

In this section, we propose an automated statistical fault localization technique (Ela) based on the statement coverage information of a test set. First, we define the methodology of Ela. Next, we briefly describe the motivating examples which are the mid() and unblock_process(ratio) functions. Finally, we discuss the results of the experiments performed on the motivating examples.

4.1. Methodology

It is assumed that the test set at hand contains at least one passed and one failed test. We use two kinds of the statement coverage of failed and passed tests: A_{ef}/A_f and A_{np}/A_p. The ratio A_{ef}/A_f stands for the frequency of a statement executed by failed tests and therefore how suspicious a statement is based on the execution information coming from the failed tests. The ratio A_{ep}/A_p stands for the frequency of a statement executed by passed tests and therefore how innocent a statement is based on the execution information coming from the passed tests. We use A_{np}/A_p which is equal to $(1 - A_{ep}/A_p)$ instead of A_{ep}/A_p in our formula. The ratio A_{np}/A_p stands for the ratio of the non-executions of a statement in passed tests and therefore how suspicious it is based on the execution information coming from the passed tests. Finally, we decided to take the geometric mean of the suspiciousness values coming from both kinds of tests as follows:

$$Suspiciousness(\mathrm{s_i}) = \left\{ \sqrt{\frac{\mathrm{A_{ef}}(s_i)}{\mathrm{A_f}} . \frac{\mathrm{A_{np}}(s_i)}{\mathrm{A_p}}} . \quad \text{if } \mathrm{A_f} \neq 0 \ \& \ \mathrm{A_p} \neq 0 \right\} \tag{12}$$

The geometric mean is a kind of mean or average which shows the central tendency of the objects being averaged by using the product of their values. It is often used for comparing different objects, each of which has multiple properties that have different numeric ranges

(Shin, Muthaiyah, & Raman, 2012). The use of a geometric mean normalizes the ranges of the properties in order to neutralize their weighting effects. A change in the values of the properties has the same effect on the geometric mean. When data consist of percentages, ratios, rates of change, the geometric mean is a useful measure of central tendency (Brase & Brase, 2011).

Since software developers try to write their best code without making any faults and software testers find most of the faults during the testing process, A_p is generally much larger than A_f. Hence; we decided to use the geometric mean for averaging the suspiciousness values coming from failed and passed tests in order to neutralize their weighting effects.

However, there are some special cases that must be considered in this formula. For example, let s_1, s_2, and s_3 be three statements Assume that s_1 is executed in m passed tests and no fail test executed it; s_2, is never executed by any test, s_3 is executed in n failed tests and no passed tests. Then, $A_{ef}(s_1)/A_f = 0$, $A_{np}(s_1)/A_p = m$, $A_{ef}(s_2)/A_f = 0$ and $A_{np}(s_2)/A_p = 0$, $A_{ef}(s_3)/A_f = n$ and $A_{np}(s_3)/A_p = 0$. Thus, suspiciousness(s_1) = sqrt(0 x m) = 0, suspiciousness(s_2) =sqrt(0 x 0) = 0, and suspiciousness(s_3) = sqrt(n x 0) = 0 which means that suspiciousness values for these three statements are all equal to 0 because of the multiplication property of zero. However, s_1 and s_3 are executed by pass and failed tests respectively. Therefore, we use y = power(2, x) power transformation function to overcome the problem in these special cases.

$$Suspiciousness(s_i) = \left\{ \sqrt{2^{\frac{A_{ef}(s_i)}{A_f}} . 2^{\frac{A_{np}(s_i)}{A_p}}} \quad \text{if } A_f \neq 0 \ \& \ A_p \neq 0 \right\} \tag{13}$$

4.1.1. The Effects of the Bases of $A_{ef}(s_i)/A_f$ and $A_{np}(s_i)/A_p$

We performed several experiments and measure the effects of the bases of $A_{ef}(s_i)/A_f$ and $A_{np}(s_i)/A_p$ on the suspiciousness value. First we set the base of $A_{np}(s_i)/A_p$ to 2 and increased the base of $A_{ef}(s_i)/A_f$ from 2 to 10 by 1 and measure their effects on the suspiciousness values. We realized that the expense is decreased for the bases in this range. Second we set the base of $A_{ef}(s_i)/A_f$ to 2 and increased the base of $A_{np}(s_i)/A_p$ from 2 to 10 and measure their effects on the suspiciousness values. We realized that the expense is increased for the bases in this range. We also measured the effects of decreasing the base of $A_{np}(s_i)/A_p$ from 2 to 1 by 0.1 on the suspiciousness values. We realized that the expense is decreased for the bases in this range. We performed the experiments on a train set which contains the 15 versions of PrintTokens and PrintTokens2 program and selected the bases of $A_{ef}(s_i)/A_f$ and $A_{np}(s_i)/A_p$ on

this set. Afterwards, we validated the selected bases of $A_{ef}(s_i)/A_f$ and $A_{np}(s_i)/A_p$ on the test set which contains 103 versions of Replace, Schedule, Schedule2, Tcas, and TotInfo programs.

Figure 2 displays that increasing the base of A_{ef}/A_f from 2 to 10 by 1 has a positive effect on the suspiciousness value on average of all the subject programs. Improvements in the expense for the bases between 4 and 10 (i.e. the circles right to the white circle) are less than 1% and can be ignored. Therefore, we decided to increase and use the base of A_{ef}/A_f as 4.

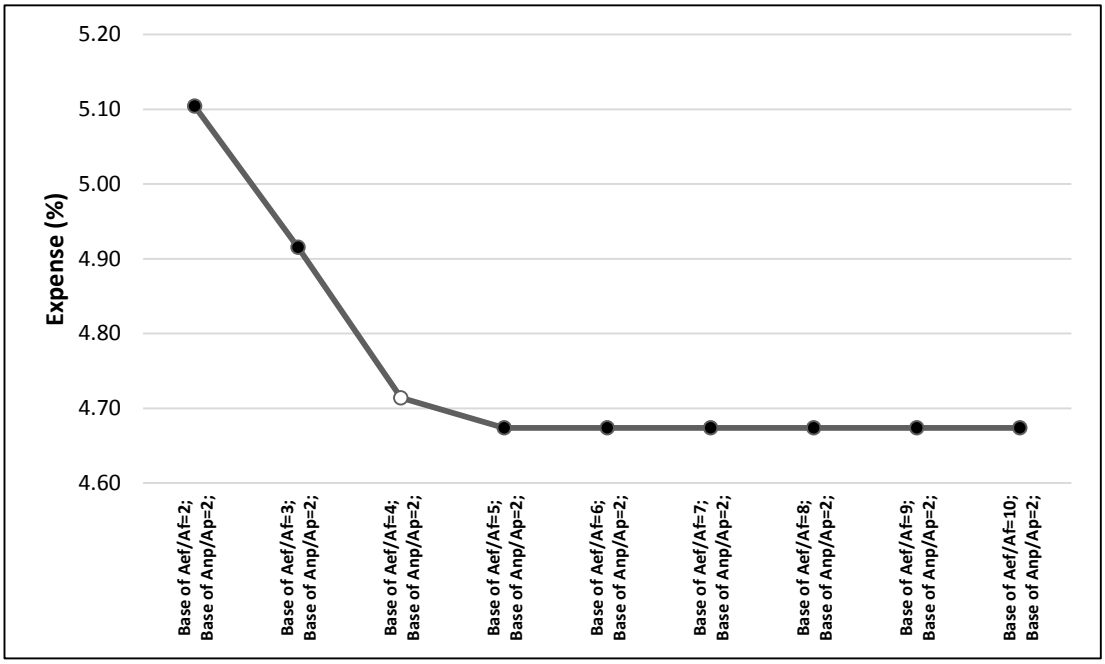

Figure 2 – Effect of the base of A_{ef}/A_f from 2 to 10

Figure 3 shows that increasing the base of A_{np}/A_p from 2 to 10 by 1 has a negative effect on the suspiciousness value on average of all the subject programs. Therefore, we decided not to increase and use the base of A_{np}/A_p and as 2.

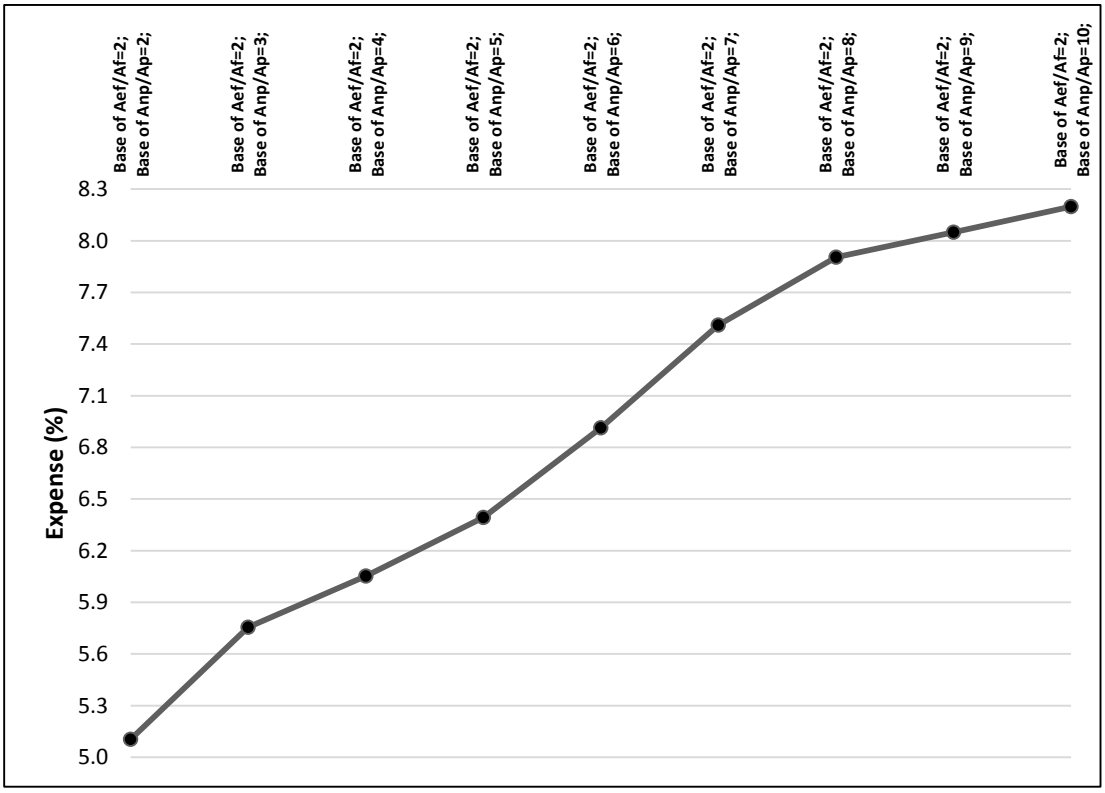

Figure 3 – Effect of the base of A_{np}/A_p from 2 to 10

Figure 4 displays that decreasing the base of A_{np}/A_p from 2 to 1 by 0.1 has a positive effect on the suspiciousness value on average of all the subject programs. Moreover, it indicates that omitting this coefficient would be a crucial mistake. The expense rises dramatically when the base of A_{np}/A_p is 1 which is equal to excluding this coefficient. Improvements in the expense for the bases between 1 and 2 (i.e. the circles left to the white circle) are less than 1% and can be ignored. Therefore, we decided to use the base of A_{np}/A_p as 2.

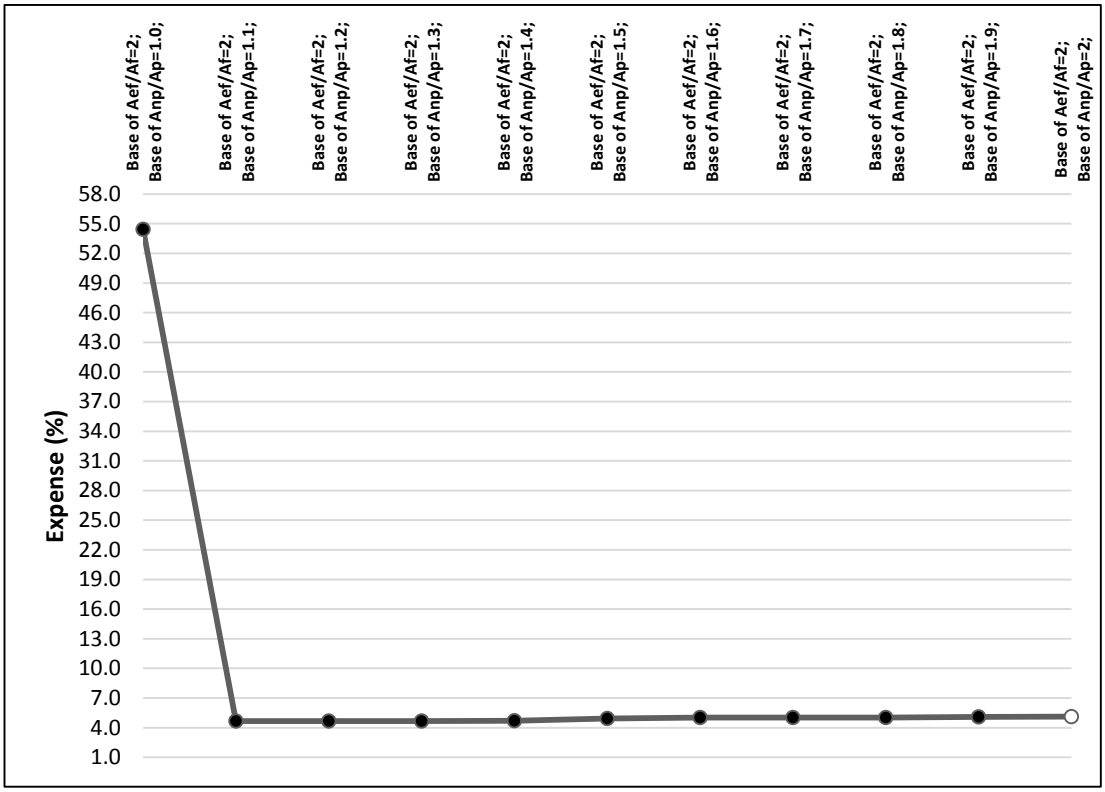

Figure 4 – Effect of the base of A_{np}/A_p between 1 and 2

There can be some cases in which the statement coverage information of failed tests are the same but the statement coverage information of passed tests are different. In these cases, the faulty statement and the innocent statements cannot be differentiated from each other. Therefore, A_{np}/A_p should be included to the calculation of suspiciousness values. A detailed discussion is given on the two motivating examples in the subsection 4.2. For example, the expense rises dramatically when the base of A_{np}/A_p is 1 which is equal to excluding this coefficient in Figure 3. Omitting this coefficient would be a crucial mistake.

Based on these empirical results, we decide to use the base of $A_{ef}(s_i)/A_f$ as 4 and the base of $A_{np}(s_i)/A_p$ as 2. The formula is updated with these bases as follows:

$$Suspiciousness(s_i) = \left\{ \sqrt{4^{\frac{A_{ef}(s_i)}{A_f}} . 2^{\frac{Anp(s_i)}{Ap}}} \quad \text{if } A_f \neq 0 \ \& \ A_p \neq 0 \right\} \tag{14}$$

4.2. Motivating Examples

We have chosen the mid() function as the first motivating example (Jones et al., 2002). It is a function that finds the middle number of three numbers inputted. At the line 7, the assignment "m=x" is mistakenly coded as the assignment "m=y". The source code of mid() function is given in Figure 5.

```
     mid() {
       int x, y, z, m;
s1     read("Enter 3 numbers: ", x, y, z);
s2     m = z;
s3     if (y<z)
s4       if (x<y)
s5         m = y;
s6       else if (x<z)
s7         m = x; // changed to: m = y
s8     else
s9       if (x>y)
s10        m = y;
s11      else if (x>z)
s12        m = x;
s13    print("Middle number is: ", m);
     }
```

Figure 5 – Source code of mid() function

There are five passed test cases and one failed test case written for mid() function. These test cases are given in Table 3. Table 3 gives the following information for these cases: name, input, expected output, actual output, and test result.

Table 3 – Five passed test cases and one failed test case written for mid() function

Test Case	Input			Expected Output	Actual Output	Test Result
	x	y	z			
T_1	3	3	4	3	3	P
T_2	2	3	4	3	3	P
T_3	4	3	2	2	2	P
T_4	1	3	2	2	2	P
T_5	5	3	4	3	3	P
T_6	3	2	4	3	2	F

The coverage matrix of the statements and the result vector of tests are given in Table 4. In this example, there is only one fail test (T_6) and five pass tests.

Table 4 – Coverage matrix and result vector of mid() function

Test	S_1	S_2	S_3	S_4	S_5	S_6	**S_7 (Fault)**	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	Result Vector
T_1	1	1	1	1	0	1	**1**	0	0	0	0	0	1	1 (P)
T_2	1	1	1	1	1	0	**0**	0	0	0	0	0	1	1 (P)
T_3	1	1	1	0	0	0	**0**	1	1	1	0	0	1	1 (P)
T_4	1	1	1	0	0	0	**0**	1	1	0	1	0	1	1 (P)
T_5	1	1	1	1	0	1	**0**	0	0	0	0	0	1	1 (P)
T_6	1	1	1	1	0	1	**1**	0	0	0	0	0	1	0 (F)

Let us exemplify how the technique is applied on the statement S_7. For this statement, based on the coverage matrix in Table 4, A_{ef}=1, A_f=1, A_{np}=4, and A_p= 5. Hence the suspiciousness value for this statement $\sqrt{4^{1/1}.2^{4/5}}$ = 2.639. We apply Ela to the motivating example and the technique assigns the suspiciousness values to the statements as in Table 5. Table 5 shows the suspiciousness values as well as the rankings (e.g. J. Rank denotes the ranking assigned by Jaccard) assigned by the SFL techniques to the statements of the program.

Table 5 – The suspiciousness values and ranks of the statements on mid() function

Statement	Jaccard	J. Rank	Tarantula	T. Rank	Ochiai	O. Rank	Ela	E. Rank
S_1	0.167	7	0.5	7	0.408	7	2	7
S_2	0.167	7	0.5	7	0.408	7	2	7
S_3	0.167	7	0.5	7	0.408	7	2	7
S_4	0.25	3	0.625	3	0.5	3	2.297	3
S_5	0	13	0	13	0	13	1.32	11
S_6	0.333	2	0.714	2	0.577	2	2.462	2
S_7 (Fault)	**0.5**	**1**	**0.833**	**1**	**0.707**	**1**	**2.639**	**1**
S_8	0	13	0	13	0	13	1.231	13
S_9	0	13	0	13	0	13	1.231	13
S_{10}	0	13	0	13	0	13	1.32	11
S_{11}	0	13	0	13	0	13	1.32	11
S_{12}	0	13	0	13	0	13	1.414	8
S_{13}	0.167	7	0.5	7	0.408	7	2	7

As Table 5 shows, Ela gives rank 1 to the faulty statement which is line 7. Actually, Ela suggests the developers examine the lines in following order: S_7, S_6, S_4, S_5, S_{10}, S_{11}, S_8, S_9, S_1, S_2, S_3, S_{13}, S_{12}. This order of the statements is generated according to their suspiciousness ranks in descending order.

As mentioned in 4.1.1, there can be some cases in which the statement coverage information of failed tests are the same but the statement coverage information of passed tests are different. For example, if we have excluded A_{np}/A_p from the calculation of suspiciousness values then A_{ef}/A_f for the failed test T_6 are the same for the following statements: S_1, S_2, S_3, S_4, S_6, S_7 (fault), and S_{13} on the first motivating example. Then the proposed technique would assign the same rank to these 7 statements including the faulty statement and could not have differentiated the faulty statement from the innocent statements. As a result, we should include A_{np}/A_p to the calculation of suspiciousness values in order to differentiate the faulty statement from the innocent statements.

We have chosen the unblock_process(ratio) function as the second motivating example (Zhang et al., 2009). It is a function that processes the queue according to its priority. At the line 6, the assignment "count=block_queue->mem_count" is mistakenly coded as the assignment "count=block_queue->mem_count+1". Five passed tests and two failed tests are written for it. The source code of unblock_process(ratio) function is given in Figure 6.

```
                void unblock_process(ratio)
                float ratio;
                {
S1                 int count;
S2                 int n;
S3                 Ele *proc;
S4                 int prio;
S5                 if (block_queue) {
S6                    count=block_queue->mem_count;
                      // count=block_queue->mem_count+1
S7                    n = (int) (count*ratio+1);
S8                    proc=find_nth(block_queue,n);
S9                    if (proc) {
S10                      block_queue=del_ele(block_queue,proc);
S11                      prio=proc->priority;
S12                      prio_queue[prio]=append_ele(prio_queue[prio],proc);
S13                   }
S14                }
                }
```

Figure 6 – Source code of unblock_process function

The coverage matrix of the statements and the result vector of tests are given in Table 6. In this example, there are two fail tests (T_3 & T_6) and five pass tests.

Table 6 – Coverage matrix and result vector of unblock_process function

Test	S_1	S_2	S_3	S_4	S_5	**S_6 (Fault)**	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	Result Vector
T_1	1	1	1	1	1	**1**	1	1	1	0	0	0	1	1	1 (P)
T_2	1	1	1	1	1	**0**	0	0	0	0	0	0	1	1	1 (P)
T_3	1	1	1	1	1	**1**	1	1	1	1	1	1	1	1	0 (F)
T_4	1	1	1	1	1	**0**	0	0	0	0	0	0	1	1	1 (P)
T_5	1	1	1	1	1	**1**	1	1	1	0	0	0	1	1	1 (P)
T_6	1	1	1	1	1	**1**	1	1	1	0	0	0	1	1	0 (F)
T_7	1	1	1	1	1	**0**	0	0	0	0	0	0	1	1	1 (P)

Let us exemplify how the technique is applied on the statement S6. For this statement, based on the coverage matrix in Table 6, A_{ef}=2, A_f=2, A_{np}=3, and A_p= 5. Hence the suspiciousness value for this statement $\sqrt{4^{2/2}.2^{3/5}}$ = 2.463. We apply Ela to the motivating example and the technique assigns the suspiciousness values to the statements as in Table 7. Table 7 shows the suspiciousness values as well as the rankings (e.g. J. Rank denotes the ranking assigned by Jaccard) assigned by the SFL techniques to the statements of the program.

Table 7 – The suspiciousness values and ranks of the statements on unblock_process function

Statement	Jaccard	J. Rank	Tarantula	T. Rank	Ochiai	O. Rank	Ela	E. Rank
S_1	0.286	14	0.5	14	0.535	14	2	14
S_2	0.286	14	0.5	14	0.535	14	2	14
S_3	0.286	14	0.5	14	0.535	14	2	14
S_4	0.286	14	0.5	14	0.535	14	2	14
S_5	0.286	14	0.5	14	0.535	14	2	14
S_6 (Fault)	**0.5**	**7**	**0.714**	**7**	**0.707**	**7**	**2.463**	**4**
S_7	0.5	7	0.714	7	0.707	7	2.463	4
S_8	0.5	7	0.714	7	0.707	7	2.463	4
S_9	0.5	7	0.714	7	0.707	7	2.463	4
S_{10}	0.5	7	1	3	0.707	7	2	14
S_{11}	0.5	7	1	3	0.707	7	2	14
S_{12}	0.5	7	1	3	0.707	7	2	14
S_{13}	0.286	14	0.5	14	0.535	14	2	14
S_{14}	0.286	14	0.5	14	0.535	14	2	14

As Table 7 shows, Ela gives rank 4 to the faulty statement which is line 6. Actually, Ela suggests the developers examine the lines in following order: S_6, S_7, S_8, S_9, S_1, S_2, S_3, S_4, S_5, S_{10}, S_{10}, S_{12}, S_{13}, S_{14}.

The second motivating example shows that Jaccard technique cannot distinguish the faulty statement from other statements, Tarantula and Ochiai techniques distinguish the faulty statement from other statements but mislead the developers to wrong statements. On the other hand, Ela technique leads the developers to the faulty statement correctly earlier than other three techniques. The intuition behind this difference is that Jaccard, Tarantula, and Ochiai techniques either do not consider $A_{np}(s_i)$ in their metrics or assign same weight to it with $A_{ef}(s_i)$ while considering it. On the other hand, Ela technique considers $A_{ef}(s_i)$ and $A_{np}(s_i)$ at the same time and assigns more appropriate weights to them. Therefore, Ela technique neutralizes the bias caused by the weights of $A_{ef}(s_i)$ and $A_{np}(s_i)$ and assigns the suspiciousness values to the statements more properly.

4.3. Discussion on the Application of Ela to the Multiple Faults

There is a traditional approach for the multiple fault localization. In this approach, faults are located and corrected one by one iteratively by the software developers until no faults remain in the program. First, tests are run on the subject program. Second, a coverage matrix is generated from the execution information in these tests. Third, a fault localization technique

is applied to this coverage matrix and the ordered list of suspicious statements is generated. Fourth, software developers locate the fault by using this list and correct it. Fifth, tests are rerun. If there is still a test that fails, then there is a fault in the program and the process starts from the beginning. Yu et al., (2008) evaluates the effectiveness of this approach on the multiple fault localization. They use SBI, Jaccard, Ochiai, and Tarantula metrics in their study. DiGiuseppe and Jones (2011) investigate the ability of the coverage based fault localization techniques to effectively localize multiple faults by suing this approach. They use Tarantula metric in their study. The proposed metric Ela could be used in this approach as the base metric since its requirements are the same as the Tarantula metric.

There are different approaches proposed for multiple fault location in the literature. Jones, Bowring, and Harrold (1995) present a parallel debugging technique which generates a specific set of test cases for each of multiple faults and assigns each set to a specific developer for simultaneous debugging of multiple faults. Tarantula technique is applied to each specific set of test cases in their research. Liu and Han (2006) propose a new type of failure proximity (R-Proximity) in order to group the failing traces due to the same fault. The failing traces are considered as similar traces if they suggest approximately the same fault location in this proximity. They suggest the fault locations according to the failing traces by using SOBER technique in their study.

Several metrics such as SBI, Jaccard, Ochiai, Tarantula, and SOBER are used in these approaches. Since the requirements and inputs of all these techniques are the same as Ela, we can use Ela as the base metric for these multiple fault localization approaches.

CHAPTER 5

PROPOSED TEST SUITE REDUCTION STRATEGIES

In this chapter, we will present four test suite reduction strategies.

5.1. Heuristic I: FminCov Test Suite Strategy

We propose a failed test reduction strategy called "FminCov Test Suite Strategy" for programs with single fault. This strategy can be extended to multiple fault programs as a heuristic. There are several approaches: Yu et al., (2008); DiGiuseppe and Jones (2011); Jones et al. (1995); Liu and Han (2006) for the programs with multiple faults. Our strategy can be applied to these approaches as a fault localization metric.

	Statements	T_1	T_2	T_3	T_4	T_5	T_6	T_7
	void unblock_process(ratio) float ratio; {							
s_1	int count;	1	1	1	1	1	1	1
s_2	int n;	1	1	1	1	1	1	1
s_3	Ele *proc;	1	1	1	1	1	1	1
s_4	int prio;	1	1	1	1	1	1	1
s_5	if (block_queue) {	1	1	1	1	1	1	1
s_6	count=block_queue->mem_count; // **count=block_queue->mem_count+1**	1	0	1	0	1	1	0
s_7	n = (int) (count*ratio+1);	1	0	1	0	1	1	0
s_8	proc=find_nth(block_queue,n);	1	0	1	0	1	1	0
s_9	if (proc) {	1	0	1	0	1	1	0
s_{10}	block_queue=del_ele(block_queue,proc);	0	0	1	0	0	0	0
s_{11}	prio=proc->priority;	0	0	1	0	0	0	0
s_{12}	prio_queue[prio]=append_ele(prio_queue[prio],proc);	0	0	1	0	0	0	0
s_{13}	}	1	1	1	1	1	1	1
s_{14}	}	1	1	1	1	1	1	1
	}							
	Test Results	1	1	0	1	1	0	1

Figure 7 – Example code segment and its coverage matrix

Let us explain the reason behind the failed test elimination with an example. Consider the code segment with corresponding coverage matrix and test result given in Figure 7. This code segment is from the program called "replace" in the Siemens suite. In this example the fault is in statement S_6. There are two failed tests, T_3 and T_6 and they have different coverage vectors. T_3 has executed all the statements that T_6 has executed plus three more statements. With this coverage matrix and the result vector, Tarantula and Ochiai reports the statement S_{10}, S_{11}, S_{12} to be the most suspicious of being faulty. Jaccard gives the same suspiciousness rank to S_6 to S_{12}. This obstruction is caused by the failed test T_3. Since there is a single fault, the additional statements that T_3 has executed are actually innocent statements. However, according to SFL approach, the suspiciousness of a statement increases with the number of failed tests that executed the statement. Therefore, the test T_3 is misleading the SFL effectiveness by causing innocent program elements to get high suspiciousness values.

We propose to eliminate the failed tests that mislead the SFL ranking, such as T_3 in the example. Since there is a single fault, all the failed tests should execute that statement. Let $cov(T)$ represent the coverage vector of the test T and $cov(T)(s)$ be a function that returns 1 if T has executed the statement s and 0 otherwise. Let T_1 and T_2 be two failed tests, and n be the number of statements. We define a subsume relation as follows: T_1 subsumes T_2 when $cov(T_1) = cov(T_2)$ and for each statement $0 < s \leq n$, $cov(T_2)(s) = 1 \Rightarrow cov(T_1)(s) = 1$. Based on this definition, we remove all the failed tests from the test suite who subsumes at least one other failed test. We could have found the minimum coverage vector that is subsumed by all the failed tests. However, we do not want to insert artificial test coverage information, therefore; we are not applying the boolean ∧ operation on the elements of the coverage vectors.

We also propose two different reduction strategies for passed tests to be used in combination with the failed test reduction strategy. They are presented in the next two subsections.

5.2. Heuristic II: FminCov Cluster Test Suite Strategy

Our goal is to find a subset of tests that have better test suite quality by clustering of failed and passed tests. We propose a strategy called "FminCov Cluster Test Suite Strategy" to obtain this quality. In this strategy, we cluster the test suite resulted from the failed test reduction (FminCov Test Suite Strategy) into subsets by using a clustering technique. After the clustering, we choose the subgroup that contains the failed test as the new reduced test suite. This reduction eliminates the passed tests that are not similar to the failed ones. The

intuition behind this novel technique is in parallel with Zeller (1999) who state that a passing run closest to a failing run contains the most information.

To implement this reduction technique, we need to decide on a clustering technique and a distance metric. Aggarwal and Reddy divide distance based clustering techniques into two types: partitioning and hierarchical (Aggarwal and Reddy, 2013). Three representatives of partitioning clustering technique are k-means, k-medians, and k-medoids. Two representatives of hierarchical clustering technique are agglomerative and divisive. Although partitioning clustering is generally faster, hierarchical clustering generates more accurate clusters. Unlike hierarchical clustering, partitioning clustering algorithms need to choose initial cluster centers which highly affect the resulting clusters. In other words, they are sensitive to the initially selected cluster centers. The partitioning clustering techniques not necessarily find the optimum clusters because of this sensitivity. Moreover, hierarchical clustering returns a much more meaningful and subjective division of clusters with the help of a dendrogram while partitioning clustering results in exactly ‘k’ clusters. Partitioning clustering is more suitable for the round shaped and roughly equal density cluster while hierarchical clustering is more suitable for the non-round shaped and different density clusters (Aggarwal and Reddy, 2013). There are different sized clusters in our data. Therefore, we prefer applying hierarchical clustering algorithms. The results of hierarchical clustering can be presented in a dendrogram. This dendrogram can be used to obtain the desired number of clusters by “cutting” it at the proper level. There are two types of hierarchical clustering: agglomerative and divisive. First one is a "bottom up" approach in which each observation starts in its own cluster, and pairs of clusters are merged as moving up the hierarchy. Second one is a "top down" approach in which all observations start in one cluster, and splits are performed recursively as moving down the hierarchy. The complexity of agglomerative clustering ($O(n^3)$) is lower than the complexity of divisive clustering ($O(2^n)$) in general cases. This makes agglomerative clustering faster than divisive clustering for large data sets. Therefore, we decide to use agglomerative clustering technique.

Agglomerative clustering technique uses various distance metrics to merge two clusters such as Euclidean, City block, Minkowski, Chebychev, Mahalanobis, Cosine, Correlation, Spearman, Hamming, and Jaccard. Since feature vectors contain binary data of the statement coverage in our case, we use Hamming distance as the distance metric.

Hamming distance is the percentage of the coordinates that differ in the feature vectors. Hamming distance equation can be represented as:

$$Hamming\ Distance(\mathrm{fv}_1, \mathrm{fv}_2) = \frac{\mathrm{A}_{10}(\mathrm{fv}_1,\mathrm{fv}_2)+\mathrm{A}_{01}(\mathrm{fv}_1,\mathrm{fv}_2)}{\mathrm{A}_{11}(\mathrm{fv}_1,\mathrm{fv}_2)+\mathrm{A}_{00}(\mathrm{fv}_1,\mathrm{fv}_2)+\mathrm{A}_{10}(\mathrm{fv}_1,\mathrm{fv}_2)+\mathrm{A}_{01}(\mathrm{fv}_1,\mathrm{fv}_2)} \quad (15)$$

$A_{11}(cov(T_1), cov(T_2))$ = Number of 1s at the same index in both coverage vectors.

$A_{00}(cov(T_1), cov(T_2))$ = Number of 0s at the same index in both coverage vectors.

$A_{10}(cov(T_1), cov(T_2))$ = Number of 1s and 0s at the same index in coverage vectors of T_1 and T_2 respectively.

$A_{01}(cov(T_1), cov(T_2))$ = Number of 0s and 1s at the same index in coverage vectors of T_1 and T_2 respectively.

Agglomerative clustering technique uses linkage criterion to decide which object to use to compute distance between clusters such as Single (shortest distance), Complete (furthest distance), Average (unweighted average distance), Weighted (weighted average distance), Centroid (centroid distance), Median (weighted center of mass distance), and Ward (inner squared distance) linkages. Centroid, Median, and Ward linkages are used when Euclidean distance metric is preferred. Since Hamming distance metric is used in this study, only single, complete, average, and weighted linkages can be used as linkage criterion. Moreover, average, and weighted linkages may not merge close groups because of outlier members that are far apart in the data.

To decide on the linkage criteria, we calculate the cophenetic correlation coefficients (Farris, 1969) for different linkage criteria when Hamming distance is used as a distance metric in agglomerative clustering. The cophenetic correlation for a cluster tree is defined as the linear correlation coefficient between the cophenetic distances obtained from the cluster tree and the original distances which are used to construct the cluster tree. In other words, it measures how realistically the cluster tree represents the dissimilarities between observations.

Table 8 – The Cophenetic Correlation Coefficients for different linkage criteria when Hamming distance is used as a distance metric

Linkage Criteria	Cophenetic Correlation Coefficient
Single	0.8026
Complete	0.8005
Average	0.8016
Weighted	0.7994

Table 8 shows that the maximum cophenetic correlation coefficient is obtained when single linkage criterion is used. Therefore, we decide to use single linkage criterion in this study.

5.2.1. Visualization of tests via Multi-Dimensional Scaling (MDS)

In our study, it is aimed to visualize where failed and passed tests are on the coordinate system according to their distance to each other based on their statement coverage information. The points are often visualized with a scatter plot. However, in some cases, the data might not be in the form of points at all, but rather in the form of pairwise similarities or dissimilarities to each other. We have multi-dimensional vectors that contain the statement coverage information of failed and passed tests in our study. The tests cannot be plotted by using their feature vectors. In these cases, it is possible to use the similarities or dissimilarities between tests to visualize where they are. The distances between the failed and tests are calculated by using the Hamming similarity metric in this study.

Multi-Dimensional Scaling (MDS) can represent the multi-dimensional data in a small number of dimensions. It does not require raw data, but only a matrix of pairwise distances or dissimilarities. The pairwise Hamming distances between tests for the first version of PrintTokens program (PrintTokens_v01) in Siemens suite are given in Table 9. Table 9 is shaded according to the Hamming distances between tests.

Table 9 – Hamming distances between tests on PrintTokens_v01 with Distinct FminCov Cluster Test Suite

	T_1	T_4	T_6	T_7	T_{10}	T_{20}	T_{21}	T_{23}	T_{24}	T_{34}	T_{45}	T_{49}	T_{55}	T_{83}	T_{90}	T_{97}	T_{271}	T_{623}	T_{1463}	T_{4101}	Result Vector
T_1	0	0.37	0.31	0.29	0.23	0.17	0.16	0.11	0.1	0.09	0.07	0.06	0.05	0.04	0.03	0.02	0.01	0.04	0.05	0.08	**P**
T_4	0.37	0	0.06	0.07	0.14	0.19	0.2	0.25	0.26	0.27	0.29	0.3	0.31	0.32	0.34	0.35	0.36	0.41	0.41	0.44	**P**
T_6	0.31	0.06	0	0.02	0.08	0.14	0.15	0.2	0.21	0.22	0.24	0.25	0.26	0.27	0.28	0.29	0.3	0.35	0.36	0.39	**P**
T_7	0.29	0.07	0.02	0	0.07	0.12	0.13	0.18	0.19	0.2	0.22	0.23	0.24	0.25	0.26	0.27	0.28	0.34	0.34	0.37	**P**
T_{10}	0.23	0.14	0.08	0.07	0	0.06	0.07	0.12	0.13	0.14	0.16	0.17	0.18	0.19	0.2	0.21	0.22	0.27	0.27	0.3	**P**
T_{20}	0.17	0.19	0.14	0.12	0.06	0	0.01	0.06	0.07	0.08	0.1	0.11	0.12	0.13	0.14	0.15	0.16	0.21	0.22	0.25	**P**
T_{21}	0.16	0.2	0.15	0.13	0.07	0.01	0	0.05	0.06	0.07	0.09	0.1	0.11	0.12	0.13	0.14	0.15	0.2	0.21	0.24	**P**
T_{23}	0.11	0.25	0.2	0.18	0.12	0.06	0.05	0	0.01	0.02	0.04	0.05	0.06	0.07	0.08	0.09	0.1	0.15	0.16	0.19	**P**
T_{24}	0.1	0.26	0.21	0.19	0.13	0.07	0.06	0.01	0	0.01	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.14	0.15	0.18	**P**
T_{34}	0.09	0.27	0.22	0.2	0.14	0.08	0.07	0.02	0.01	0	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.13	0.14	0.17	**P**
T_{45}	0.07	0.29	0.24	0.22	0.16	0.1	0.09	0.04	0.03	0.02	0	0.01	0.02	0.03	0.04	0.05	0.06	0.11	0.12	0.15	**P**
T_{49}	0.06	0.3	0.25	0.23	0.17	0.11	0.1	0.05	0.04	0.03	0.01	0	0.01	0.02	0.03	0.04	0.05	0.1	0.11	0.14	**P**
T_{55}	0.05	0.31	0.26	0.24	0.18	0.12	0.11	0.06	0.05	0.04	0.02	0.01	0	0.01	0.02	0.03	0.04	0.09	0.1	0.13	**P**
T_{83}	0.04	0.32	0.27	0.25	0.19	0.13	0.12	0.07	0.06	0.05	0.03	0.02	0.01	0	0.01	0.02	0.03	0.08	0.09	0.12	**P**
T_{90}	0.03	0.34	0.28	0.26	0.2	0.14	0.13	0.08	0.07	0.06	0.04	0.03	0.02	0.01	0	0.01	0.02	0.07	0.08	0.11	**P**
T_{97}	0.02	0.35	0.29	0.27	0.21	0.15	0.14	0.09	0.08	0.07	0.05	0.04	0.03	0.02	0.01	0	0.01	0.06	0.07	0.1	**P**
T_{271}	0.01	0.36	0.3	0.28	0.22	0.16	0.15	0.1	0.09	0.08	0.06	0.05	0.04	0.03	0.02	0.01	0	0.05	0.06	0.09	**F**
T_{623}	0.04	0.41	0.35	0.34	0.27	0.21	0.2	0.15	0.14	0.13	0.11	0.1	0.09	0.08	0.07	0.06	0.05	0	0.01	0.04	**P**
T_{1463}	0.05	0.41	0.36	0.34	0.27	0.22	0.21	0.16	0.15	0.14	0.12	0.11	0.1	0.09	0.08	0.07	0.06	0.01	0	0.03	**P**
T_{4101}	0.08	0.44	0.39	0.37	0.3	0.25	0.24	0.19	0.18	0.17	0.15	0.14	0.13	0.12	0.11	0.1	0.09	0.04	0.03	0	**P**
Result Vector	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**F**	**P**	**P**	**P**	

By using the MDS approach, the tests are plotted to 2D coordinate system based on the Hamming distance between them in Figure 8. The labels on the dots are ids of the tests in this figure.

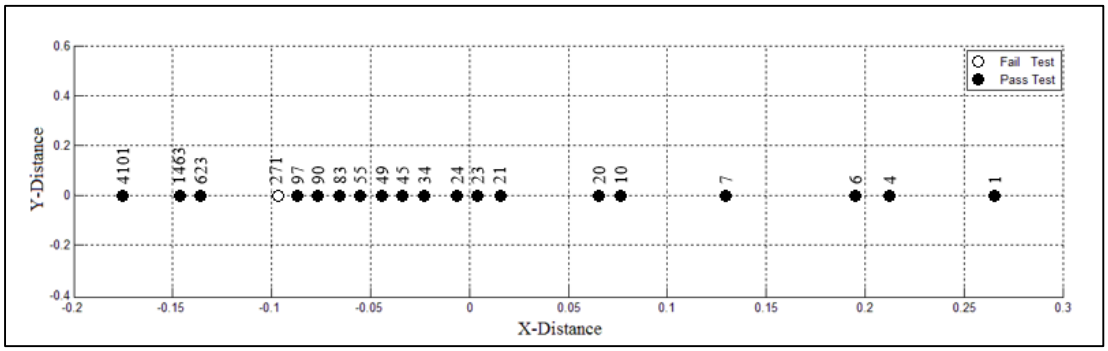

Figure 8 – Failed and pass tests before clustering of failed and passed tests on PrintTokens_v01

Figure 8 indicates that there are 1 failed test and 19 passed tests on the coordinate system. We cluster 20 tests into two clusters. We apply silhouette analysis to determine the optimum number of clusters. We cluster the tests into 2 to 5 clusters according to their feature vectors. Silhouette coefficients for 2 to 5 clusters on PrintTokens_v01 are given in Figure 9, Figure 10, Figure 11, and Figure 12.

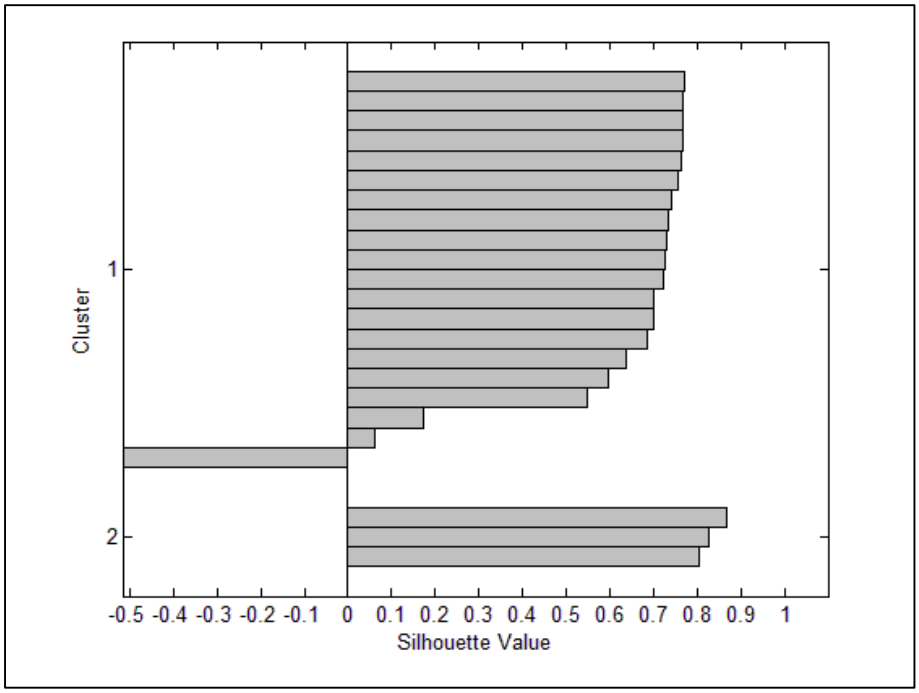

Figure 9 – Silhouette coefficients for 2 clusters on PrintTokens_v01

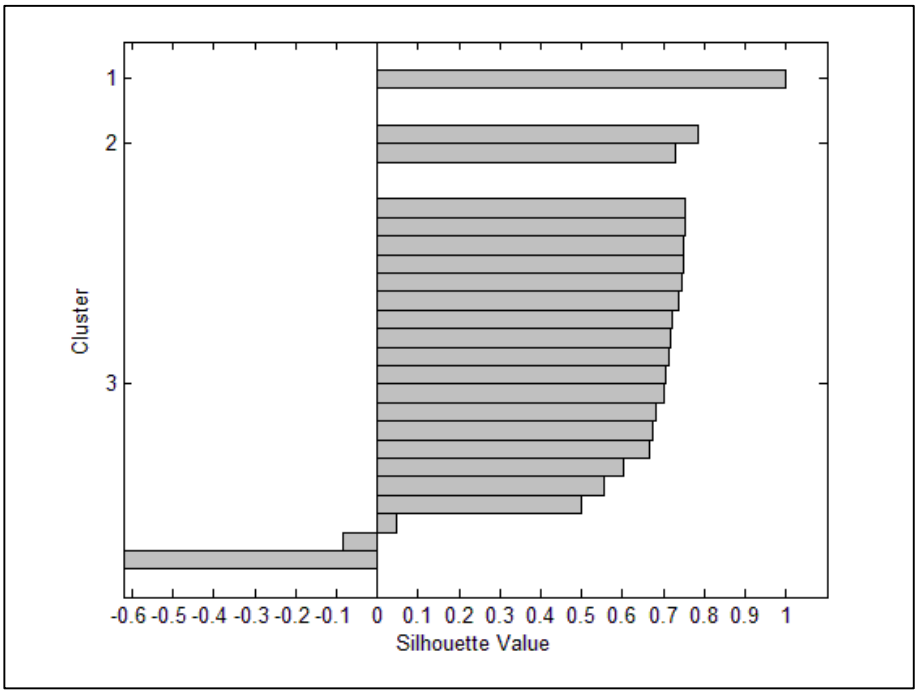

Figure 10 – Silhouette coefficients for 3 clusters on PrintTokens_v01

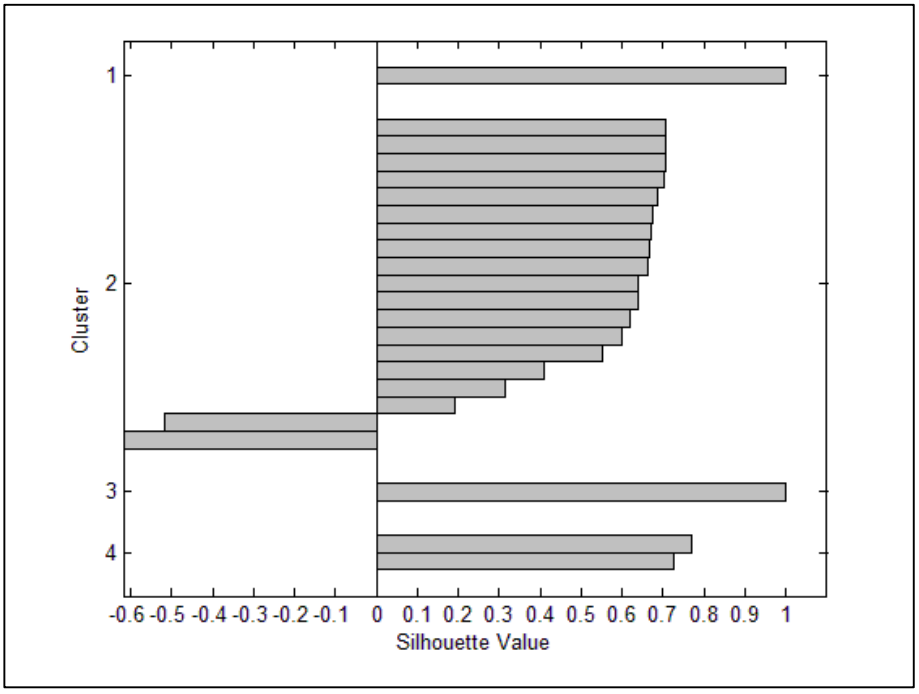

Figure 11 – Silhouette coefficients for 4 clusters on PrintTokens_v01

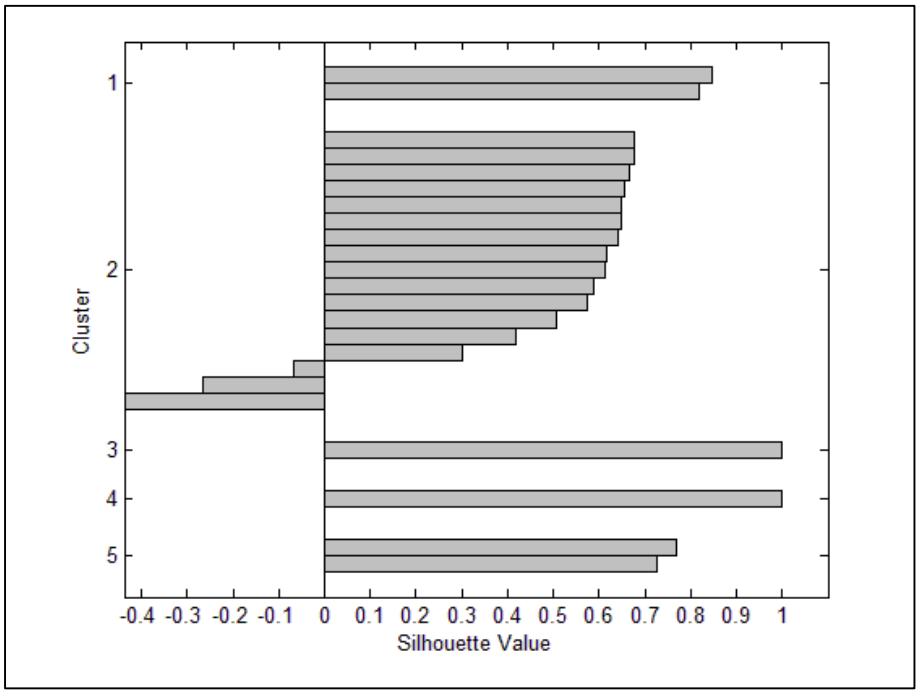

Figure 12 – Silhouette coefficients for 5 clusters on PrintTokens_v01

Our aim is to find the optimal ‘k’ value for the clustering algorithm. Therefore, we plot the mean silhouette coefficient scores for 2 to 5 clusters on PrintTokens_v01 in order to find the optimal ‘k’ value. Figure 13 shows that the optimal ‘k’ value is 2 since it has highest mean silhouette coefficient scores. Therefore, we decide to cluster the tests into 2 clusters.

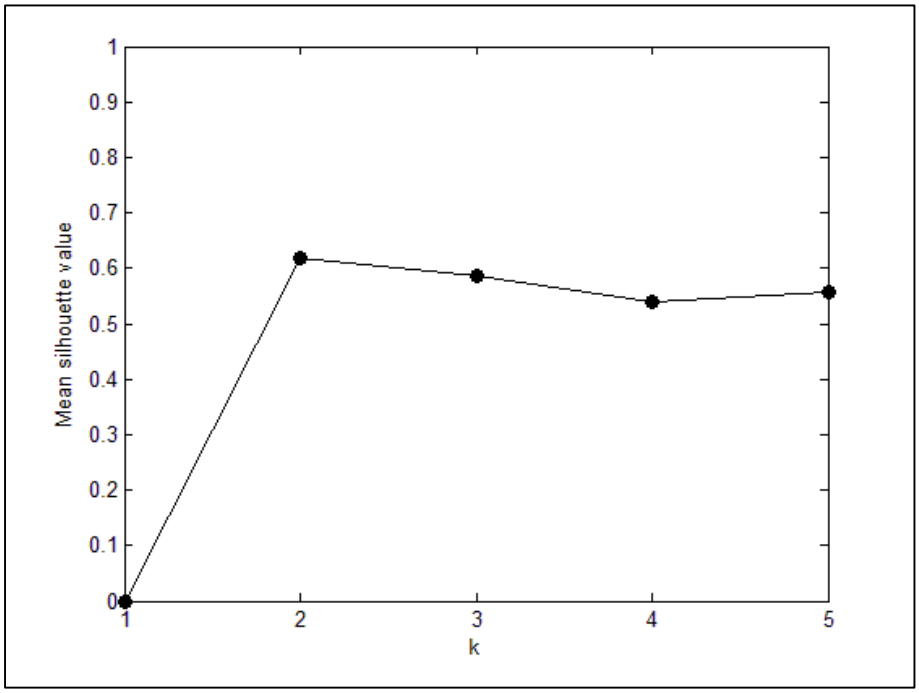

Figure 13 – Selection of optimum ‘k’ according to the mean silhouette values

We aim to find the passed tests that are not very close to the remaining failed tests and eliminate them to achieve better fault localization. Therefore, we cluster the tests into 2 cluster and eliminate the cluster that does not contain the remaining failed tests.

After completing the clustering of the test cases of PrintTokens_v01, 16 passed tests and one failed test shaped as circle are clustered together. In addition, 3 passed tests shaped as square are clustered to together. Figure 14 shows the tests and their clusters on the coordinate system. The labels on the dots are ids of the tests in this figure.

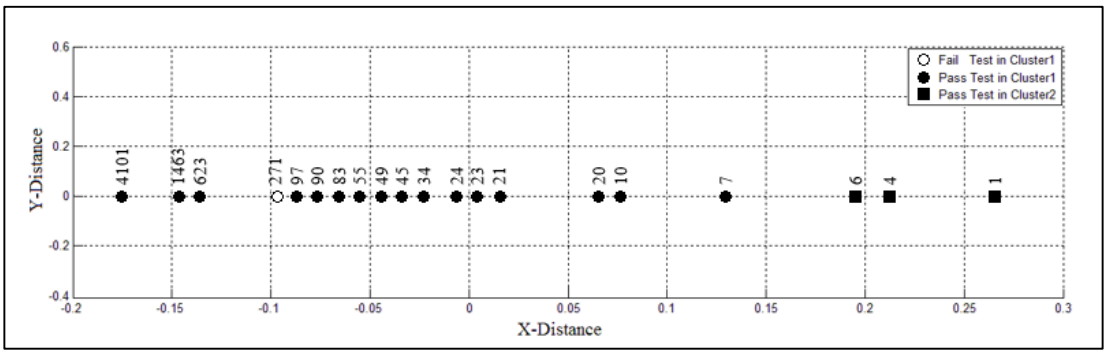

Figure 14 – Failed and pass tests after clustering of failed and passed tests on PrintTokens_v01

In order to validate agglomerative clustering visually, we use a dendrogram of tests on PrintTokens_v01. Figure 15 shows this dendrogram which contains 1 failed test and 19 passed tests. It is observed that 3 tests (thick lines) are close to each other while 17 tests (thin lines) are close to each other.

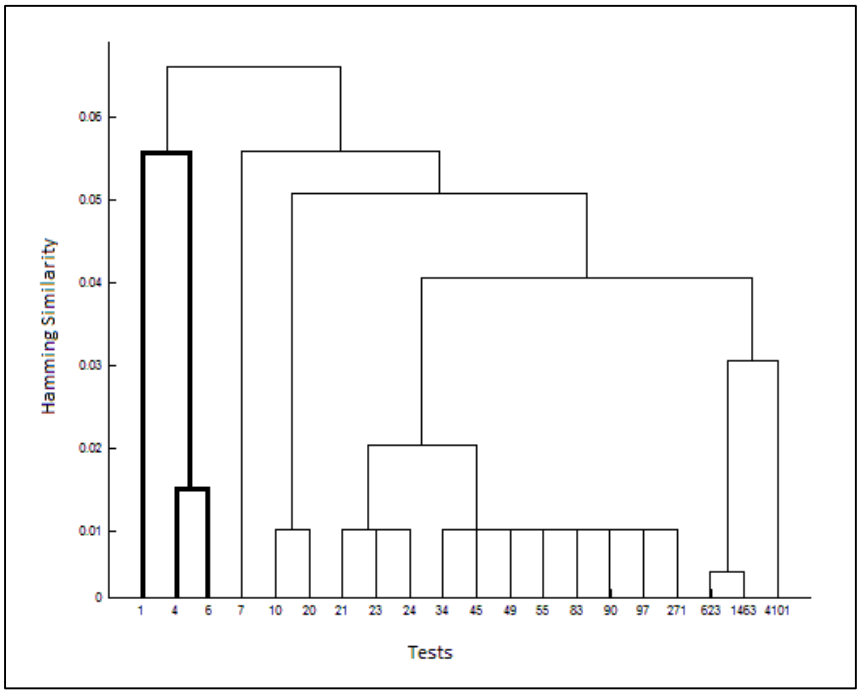

Figure 15 – Dendrogram of tests on PrintTokens_v01 [Clustering]

Our purpose is to observe the effect of Clustering strategy on the fault localization. Therefore, we evaluate the effect of FminCov Cluster Test Suite Strategy on PrintTokens_v01. Figure 16 shows this effect on PrintTokens_v01. It is observed that Clustering strategy has a decline on the fault localization.

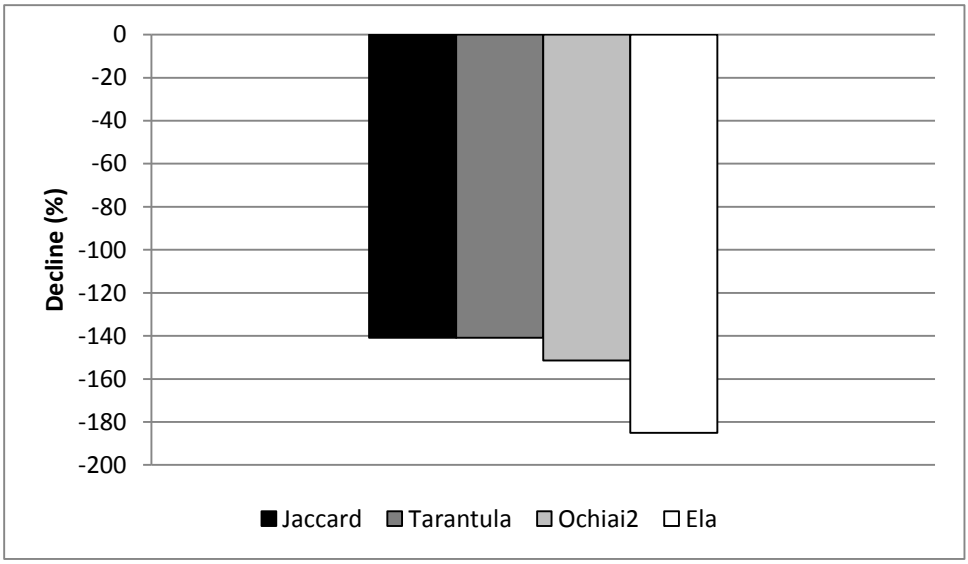

Figure 16 – The effect of FminCov Cluster Test Suite Strategy on PrintTokens_v01

5.3. Heuristic III: FminCov Classify Test Suite Strategy

Our goal is to find a subset of tests that have better test suite quality by clustering of failed and passed tests. We propose a strategy called “FminCov Cluster Test Suite Strategy” to

obtain this quality. In this strategy, we classify the passed tests into two classes of the failed tests. The eliminated failed tests (FminCov Test Suite Strategy) form one class, and the remaining failed tests form the other class. Then we remove the subgroup that contains the eliminated failed tests. This elimination removes a subset of the tests which are similar to all failed ones due to the subsumption relation among all the failed tests. The intuition is to approximate and reduce the tests that have executed the faulty statement but still pass.

There are different types of classification techniques to be used for this process: K nearest neighbors (KNN), Naïve Bayes, Support Vector Machines (SVMs), Decision Trees (DTs). KNN classification algorithm is non-parametric algorithm since it does not make any assumption on the distribution of the underlying data. Therefore, it can be used for classification when there is little or no prior knowledge about the distribution of the underlying data. Moreover, KNN classification algorithm is a lazy learning algorithm since there is no explicit model training phase and a model construction is deferred until it is needed to make a classification.

We prefer to use KNN classification algorithm since it is a simple, non-parametric and lazy learning algorithm. KNN classification algorithm uses various distance metrics such as Euclidean, City block, Minkowski, Chebychev, Mahalanobis, Cosine, Correlation, Spearman, Hamming, and Jaccard. Since feature vectors contain binary data of the statement coverage in our case, we use Hamming distance as the distance metric. We select ‘k’ as 1 which means that one nearest neighbor in each class is used to classify a new test into the classes. Therefore, we can say that we use “nearest neighbor” algorithm instead of “k nearest neighbors” algorithm.

In this strategy, we first form two classes of failed tests. The eliminated failed tests using FminCov Test Suite Strategy form one class, and the remaining failed tests form the other class. Then we classify the passed tests into appropriate classes by using KNN classification algorithm. If a passed test has the same smallest similarity to the classes of the eliminated failed tests and the remaining failed tests, then it is classified to the remaining failed tests. Finally, we remove the subgroup that contains the eliminated failed tests.

5.3.1. Visualization of tests via Multi-Dimensional Scaling (MDS)

We use MDS to visualize the data and the result of the FminCov Classify Test Suite Strategy. It is also used for visual validation of the classification process. Consider pairwise Hamming distances between tests for the first version of PrintTokens program

(PrintTokens_v01) in Siemens suite are given in Table 10. Table 10 is shaded according to the Hamming distances between tests.

Table 10 – Hamming distances between tests on PrintTokens_v01 with Distinct FminCov Classify Test Suite

	T_1	T_4	T_6	T_7	T_{10}	T_{11}	T_{21}	T_{23}	T_{24}	T_{34}	T_{45}	T_{49}	T_{55}	T_{83}	T_{90}	T_{97}	T_{271}	T_{623}	T_{1463}	T_{2592}	T_{3296}	T_{4101}	Result Vector
T_1	0	0.37	0.31	0.29	0.23	0.17	0.16	0.11	0.1	0.09	0.07	0.06	0.05	0.04	0.03	0.02	0.01	0.05	0.07	0.04	0.05	0.08	**P**
T_4	0.37	0	0.06	0.07	0.14	0.19	0.2	0.25	0.26	0.27	0.29	0.3	0.31	0.32	0.34	0.35	0.36	0.42	0.43	0.41	0.41	0.44	**P**
T_6	0.31	0.06	0	0.02	0.08	0.14	0.15	0.2	0.21	0.22	0.24	0.25	0.26	0.27	0.28	0.29	0.3	0.36	0.38	0.35	0.36	0.39	**P**
T_7	0.29	0.07	0.02	0	0.07	0.12	0.13	0.18	0.19	0.2	0.22	0.23	0.24	0.25	0.26	0.27	0.28	0.35	0.36	0.34	0.34	0.37	**P**
T_{10}	0.23	0.14	0.08	0.07	0	0.06	0.07	0.12	0.13	0.14	0.16	0.17	0.18	0.19	0.2	0.21	0.22	0.28	0.29	0.27	0.27	0.3	**P**
T_{20}	0.17	0.19	0.14	0.12	0.06	0	0.01	0.06	0.07	0.08	0.1	0.11	0.12	0.13	0.14	0.15	0.16	0.22	0.24	0.21	0.22	0.25	**P**
T_{21}	0.16	0.2	0.15	0.13	0.07	0.01	0	0.05	0.06	0.07	0.09	0.1	0.11	0.12	0.13	0.14	0.15	0.21	0.23	0.2	0.21	0.24	**P**
T_{23}	0.11	0.25	0.2	0.18	0.12	0.06	0.05	0	0.01	0.02	0.04	0.05	0.06	0.07	0.08	0.09	0.1	0.16	0.18	0.15	0.16	0.19	**P**
T_{24}	0.1	0.26	0.21	0.19	0.13	0.07	0.06	0.01	0	0.01	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.15	0.17	0.14	0.15	0.18	**P**
T_{34}	0.09	0.27	0.22	0.2	0.14	0.08	0.07	0.02	0.01	0	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.14	0.16	0.13	0.14	0.17	**P**
T_{45}	0.07	0.29	0.24	0.22	0.16	0.1	0.09	0.04	0.03	0.02	0	0.01	0.02	0.03	0.04	0.05	0.06	0.12	0.14	0.11	0.12	0.15	**P**
T_{49}	0.06	0.3	0.25	0.23	0.17	0.11	0.1	0.05	0.04	0.03	0.01	0	0.01	0.02	0.03	0.04	0.05	0.11	0.13	0.1	0.11	0.14	**P**
T_{55}	0.05	0.31	0.26	0.24	0.18	0.12	0.11	0.06	0.05	0.04	0.02	0.01	0	0.01	0.02	0.03	0.04	0.1	0.12	0.09	0.1	0.13	**P**
T_{83}	0.04	0.32	0.27	0.25	0.19	0.13	0.12	0.07	0.06	0.05	0.03	0.02	0.01	0	0.01	0.02	0.03	0.09	0.11	0.08	0.09	0.12	**P**
T_{90}	0.03	0.34	0.28	0.26	0.2	0.14	0.13	0.08	0.07	0.06	0.04	0.03	0.02	0.01	0	0.01	0.02	0.08	0.1	0.07	0.08	0.11	**P**
T_{97}	0.02	0.35	0.29	0.27	0.21	0.15	0.14	0.09	0.08	0.07	0.05	0.04	0.03	0.02	0.01	0	0.01	0.07	0.09	0.06	0.07	0.1	**P**
T_{271}	0.01	0.36	0.3	0.28	0.22	0.16	0.15	0.1	0.09	0.08	0.06	0.05	0.04	0.03	0.02	0.01	0	0.06	0.08	0.05	0.06	0.09	**F**
T_{623}	0.05	0.42	0.36	0.35	0.28	0.22	0.21	0.16	0.15	0.14	0.12	0.11	0.1	0.09	0.08	0.07	0.06	0	0.02	0.01	0.01	0.03	**P**
T_{1463}	0.07	0.43	0.38	0.36	0.29	0.24	0.23	0.18	0.17	0.16	0.14	0.13	0.12	0.11	0.1	0.09	0.08	0.02	0	0.03	0.02	0.01	**P**
T_{2592}	0.04	0.41	0.35	0.34	0.27	0.21	0.2	0.15	0.14	0.13	0.11	0.1	0.09	0.08	0.07	0.06	0.05	0.01	0.03	0	0.01	0.04	**F**
T_{3296}	0.05	0.41	0.36	0.34	0.27	0.22	0.21	0.16	0.15	0.14	0.12	0.11	0.1	0.09	0.08	0.07	0.06	0.01	0.02	0.01	0	0.03	**F**
T_{4101}	0.08	0.44	0.39	0.37	0.3	0.25	0.24	0.19	0.18	0.17	0.15	0.14	0.13	0.12	0.11	0.1	0.09	0.03	0.01	0.04	0.03	0	**P**
Result Vector	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**F**	**P**	**P**	**F**	**F**	**P**	

By using the MDS approach, the tests are plotted to 2D coordinate system based on the Hamming distance between them in Figure 17. The labels on the dots are ids of the tests in this figure.

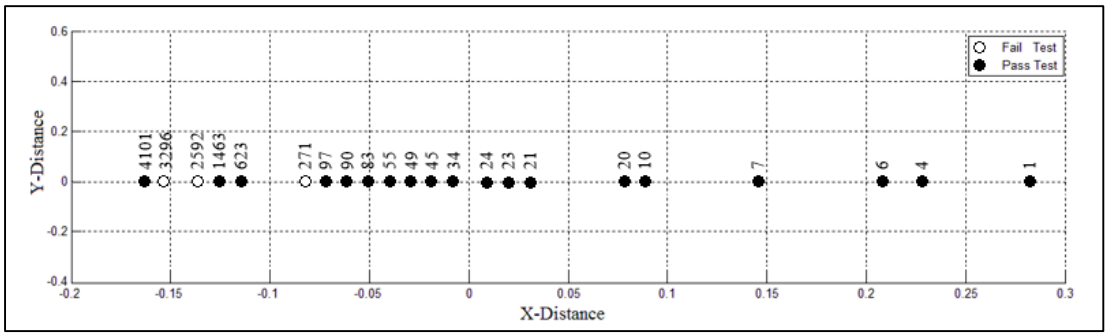

Figure 17 – Failed and pass tests before classifying the passed tests to the failed tests on PrintTokens_v01

Figure 17 indicates that there are 3 failed tests and 19 passed tests on the coordinate system. We classify 19 passed tests cases into the appropriate classes by using KNN classification algorithm.

After completing the classification process, 16 passed tests shaped as circle are classified to one failed test shaped as circle. In addition, 3 passed tests shaped as square are classified to 2 failed tests shaped as square. Figure 18 shows the tests and their classes on the coordinate system. The labels on the dots are ids of the tests in this figure.

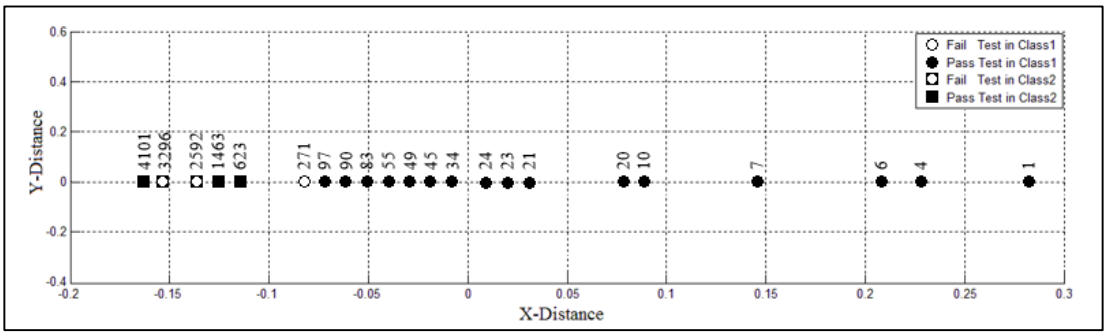

Figure 18 – Failed and pass tests after classifying the passed tests to the failed tests on PrintTokens_v01

In order to validate KNN classification visually, we use a dendrogram of tests on PrintTokens_v01. Figure 19 shows this dendrogram which contains 3 failed tests and 19 passed tests. It is observed that 5 tests (thick lines) are close to each other while 17 tests (thin lines) are close to each other.

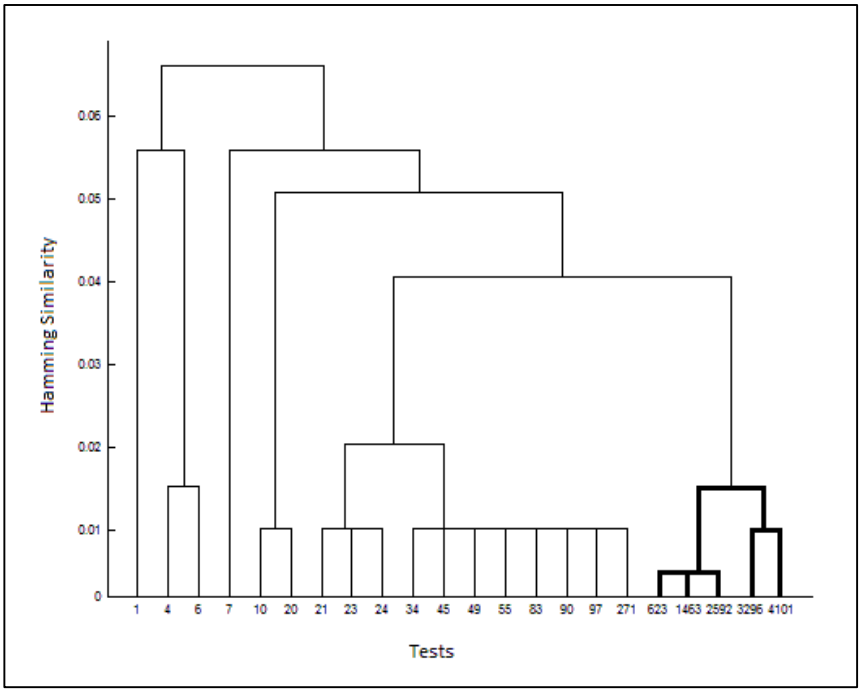

Figure 19 – Dendrogram of tests on PrintTokens_v01 [Classification]

We would like to observe the effect of Classification strategy on the fault localization. Therefore, we evaluate the effect of FminCov Classify Test Suite Strategy on

PrintTokens_v01. Figure 20 shows this effect on the PrintTokens_v01. It is observed that Classification strategy has an improvement over the fault localization.

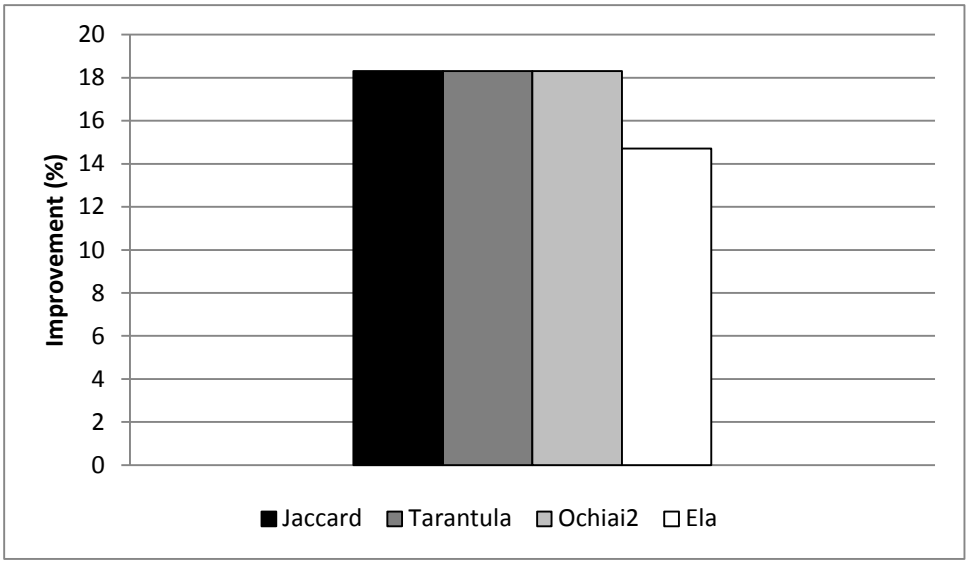

Figure 20 – The effect of FminCov Classify Test Suite Strategy on PrintTokens_v01

5.4. Heuristic IV: Equivalent Test Elimination (Distinct Test Selection)

If two tests have executed the set of same statements (i.e. if they have covered the same set of statements), then we consider them as equivalent tests with respect to their statement coverage. We claim that existence of equivalent tests affects the computation of the suspiciousness values of the statements. Existence of equivalent tests lead to the repetition of test vectors in a coverage matrix. Since the fault localization metrics are based on the number of occurrences of 1's and 0's in this coverage matrix, the repetition of test vectors will have a direct effect on the computation of the suspiciousness values. We suggest the elimination of equivalent tests from both passed and failed tests. This approach is called "Distinct Test Suite Strategy".

We categorize failed tests as good failed tests and bad failed tests. While a good failed test is defined as the failed test that has the minimum statement coverage, a bad failed test is defined as the failed test that have the greater statement coverage than the good test in this study. In the rest of this section, we analyze the effect of having equivalent tests in both of these categories.

Having more than one test with the same coverage vector in the coverage matrix corresponds to row scaling (Tan, Kumar, & Srivastava, 2002). If the SFL measures are not invariant under row scaling then the redundancy can cause an unpredictable effect on the effectiveness

of fault localization techniques. Let us show that how this redundancy affects the effectiveness of fault localization techniques.

We apply row scaling invariance property to the coverage matrix in our case. First, we replicate the good failed tests by applying scaling factor as 2. That is, we replicate all the rows of coverage matrix for good failed tests by 2. Afterwards, we check whether Jaccard, Tarantula, Ochiai, and Ela metrics are invariant or not under this process. Second, we replicate the bad failed tests by applying scaling factor as 2 and then check whether these four metrics are invariant or not under this process. We empirically see that each type of the tests has different effect on the effectiveness of fault localization techniques. Replication of good failed tests increases the effectiveness of fault localization while replication of bad tests has a negative impact as expected. Since it is not known whether a test is good test or not, we select the safe side and eliminate the equivalent tests from both failed and passed tests in the experiments.

Let us show how the effect of equivalent tests on first version of PrintTokens program (PrintTokens_v01) and how it can cause a bias on the effect of the fault localization techniques by applying the row scaling invariance property.

FailMinCov tests which are kinds of good failed tests are replicated by 2 and the results in Table 11 are obtained on the Redundant Test Suite.

Table 11 – Improvements on Redundant Test Suite where FailMinCov tests are replicated by 2

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	0%	0%	2.3%	9.92%
PrintTokens2	5.95%	7.56%	6.91%	0%
Replace	4.25%	4.04%	1.48%	2.56%
Schedule	0%	0%	0%	0%
Schedule2	0%	0%	0%	0%
Tcas	0%	0%	0%	0%
TotInfo	0.15%	0.29%	0%	0%

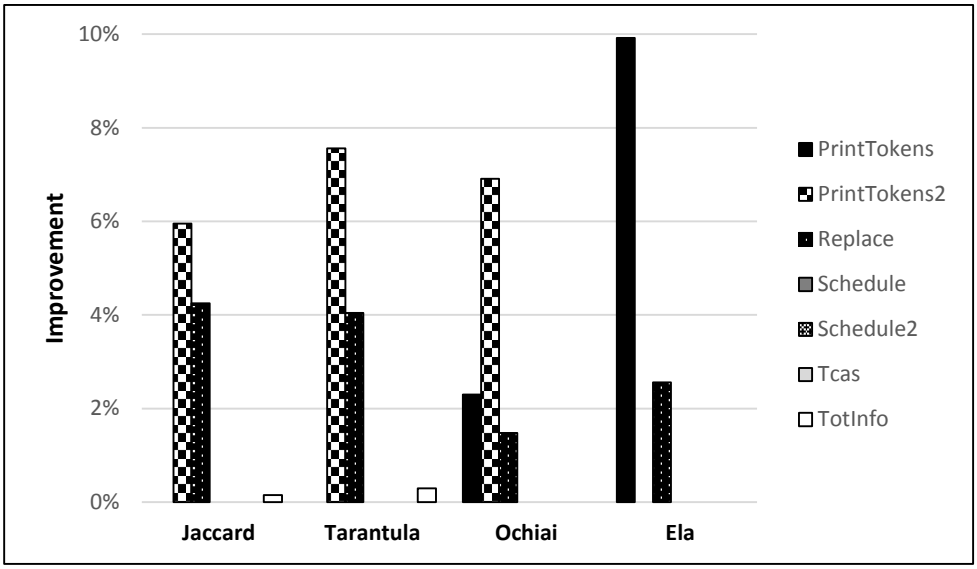

Figure 21 – Improvements by replicating the FailMinCov tests by 2

Figure 21 shows the improvements by replicating the FailMinCov tests by 2. While Ela technique has a maximum 9.92% improvement, Jaccard technique has a maximum 5.95% improvement, Tarantula technique has a maximum 7.56% improvement, and Ochiai technique has a maximum 6.91% improvement.

FailOther tests which are kinds of bad failed tests are replicated by 2 and the results in Table 12 are obtained on the Redundant Test Suite.

Table 12 – Declines on Redundant Test Suite where FailOther tests are replicated by 2

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	-22.54%	-22.54%	0%	0%
PrintTokens2	0%	-18%	-16.81%	-16.53%
Replace	-15.76%	-22.91%	-5.04%	-4.94%
Schedule	-7.45%	-16.82%	0%	0%
Schedule2	0%	0%	0%	0%
Tcas	-0.1%	-0.1%	0%	0%
TotInfo	-0.72%	-1.36%	-0.15%	-0.31%

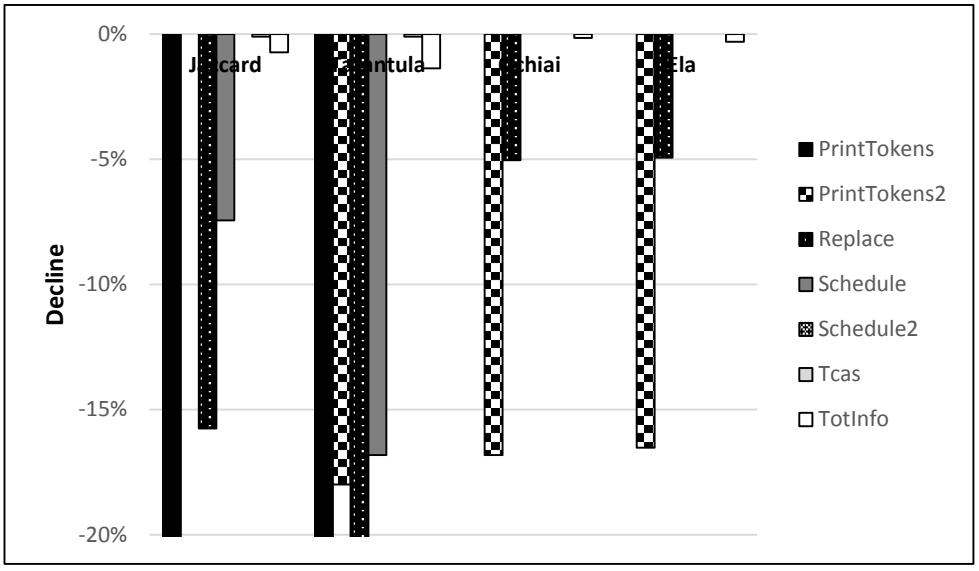

Figure 22 – Declines by replicating the FailOther tests by 2

Figure 22 shows the declines by replicating the FailOther tests by 2. While Ela technique has a maximum 16.53% decline, Jaccard technique has a maximum 22.54% decline, Tarantula technique has a maximum 22.91% decline, and Ochiai technique has a maximum 16.81% decline.

5.4.1. Discussion

Figure 21 and Figure 22 show the effects of replicating the FailMinCov tests and the FailOther tests by 2 on the expenses of PrintTokens_v01.

We observe that while replicating the FailMinCov tests by 2 has a positive effect, replicating the FailOther tests by 2 has a negative effect on the expenses of PrintTokens_v01.

Good failed tests increase the suspiciousness values of the faulty statements and the rank of faulty line is automatically decreased. Therefore, the expense for finding the faulty statement is automatically decreased too. If the good failed tests are duplicated without being aware of it, then this positive effect is duplicated too. Bad fail tests increase the suspiciousness values of the innocent statements and the rank of faulty line is automatically increased. Therefore, the expense for finding the faulty statement is automatically increased too. If the bad failed tests are duplicated without being aware of it, then this negative effect is duplicated too. Since it is not known that whether a test is good test or not, the existence of the equivalent tests can cause a bias in the ranking of statements according to their suspiciousness values. Therefore, we select the safe side and eliminate the equivalent tests from both passed and failed tests in the experiments.

CHAPTER 6

EFFECTIVE RANKING STRATEGY: LOCAL MAXIMA

In this chapter, we propose a Local Maxima strategy to obtain effective ranking of statements according to their suspiciousness values. The rationale behind this strategy is explained in the methodology section.

6.1. Methodology

Statistical fault localization techniques assign suspiciousness values to the statements according to whether they are executed or not in the tests. They aim to assign the highest suspiciousness value to the faulty statement. Moreover, they assign higher suspiciousness values to the statements near to the faulty statement since these statements are on the same execution path with the faulty statement. However, the statements near the faulty statement are not actually suspicious statements. In other words, these statements are assigned with higher suspiciousness values since they are in the neighborhood of the faulty statement. Therefore, they should be removed from the list of suspicious statements.

We propose a Local Maxima strategy to remove these statements. In this strategy, the most suspicious statements are assumed to have the highest suspiciousness values in their neighborhoods. In other words, they are assumed to be local maxima in their neighborhoods. Therefore, only the statements which are local maxima are left and the other statements are removed from the list of suspicious statements when presented to the user. We write a function called findLocalMaxima to find select the most suspicious statements which are local maxima in their neighborhoods. The algorithm of the findLocalMaxima is defined as follows:

```
/**************************************************************************
Name        :  findLocalMaxima
Function    :  This function finds the list of the local maxima values
               among the suspiciousness values of the statements of a
               program
Algorithm   :  For each statement of a program, the local maximum
               value among the suspiciousness values of its
               preceding statement, itself, and its succeeding statement
               is found and added to the list of the local maxima values
Input       :  The list of the suspiciousness values of the statements
               of a program
Output      :  The list of the local maxima values among the
               suspiciousness values of the statements of a program
**************************************************************************/
List<LocalMaximumOfStatement>
findLocalMaxima(List<SuspiciousnessOfStatement> svsList) {
  List< LocalMaximumOfStatement> lmsList = ∅;
  index := 0;
  For each statement si in svsList
    si,left := si-1; // the statement in the previous line of si
    si,right := si+1; // the statement in the next line of si
    If si ≥ si,left && si ≥ si,right Then
      LocalMaximumOfStatement lmsElement = si;
      lmsList.put(index, lmsElement);
      index++;
    End If
  End For
  Return lmsList;
}
```

Figure 23 – Pseudo code of the findLocalMaxima algorithm

In this algorithm, the list of statements whose suspiciousness values are assigned by the statistical fault localization techniques are taken as an input. Each statement in the list is compared with the statement in its previous line (left statement) and the statement in its next line (right statement). If the suspiciousness value of a statement is greater or equal to the suspiciousness values of its left and right statements at the same time, then it is added to the list of suspicious statements which will be presented to the user. At the end, the list of most suspicious statements which are local maxima in their neighborhoods are returned as an output.

6.2. Motivating Example

The suspiciousness values of all the statement of the first version of PrintTokens program (PrintTokens_v01) in the Siemens suite are given as line graphs for Jaccard, Tarantula, Ochiai, and Ela techniques in Figure 24, Figure 25, Figure 26, and Figure 27. In these figures, the x axis shows the statement number as it occurs in the program and the y axis shows the suspiciousness values assigned to the corresponding statements.

The peak statements shaped as black triangle in the line graphs are the local maxima statements since they have highest suspiciousness values in their neighborhoods. The faulty statement surrounded by a gray square is one of the local maxima statements. Notice that some statements shaped as white circles have higher suspiciousness values than the faulty statement. The faulty statement is going to be ranked after these statements. We claim that these statements are assigned with higher suspiciousness values since they are neighboring statements of local maxima statements due to their control flow. Therefore, our strategy excludes such neighboring statements and presents the user only the local maxima statements to examine instead of the whole ranked list of statements. By using this approach, the neighboring statements which have higher suspiciousness values than the faulty statement are removed and the rank of faulty statement is automatically decreased. The horizontal gray line is the threshold value for the neighboring statements which are non-local maxima statements that affect the rank of the faulty statement.

The Local Maxima strategy assumes that the faulty statement must have the greatest suspiciousness value. Therefore, the most suspicious statements must also have the greatest suspiciousness values around their neighborhoods. We examine 1-nearest neighborhoods (its left statements and its right statements) of the statements and find the local maxima statements among them. Afterwards, we present a ranked list of these local maxima statements to the user.

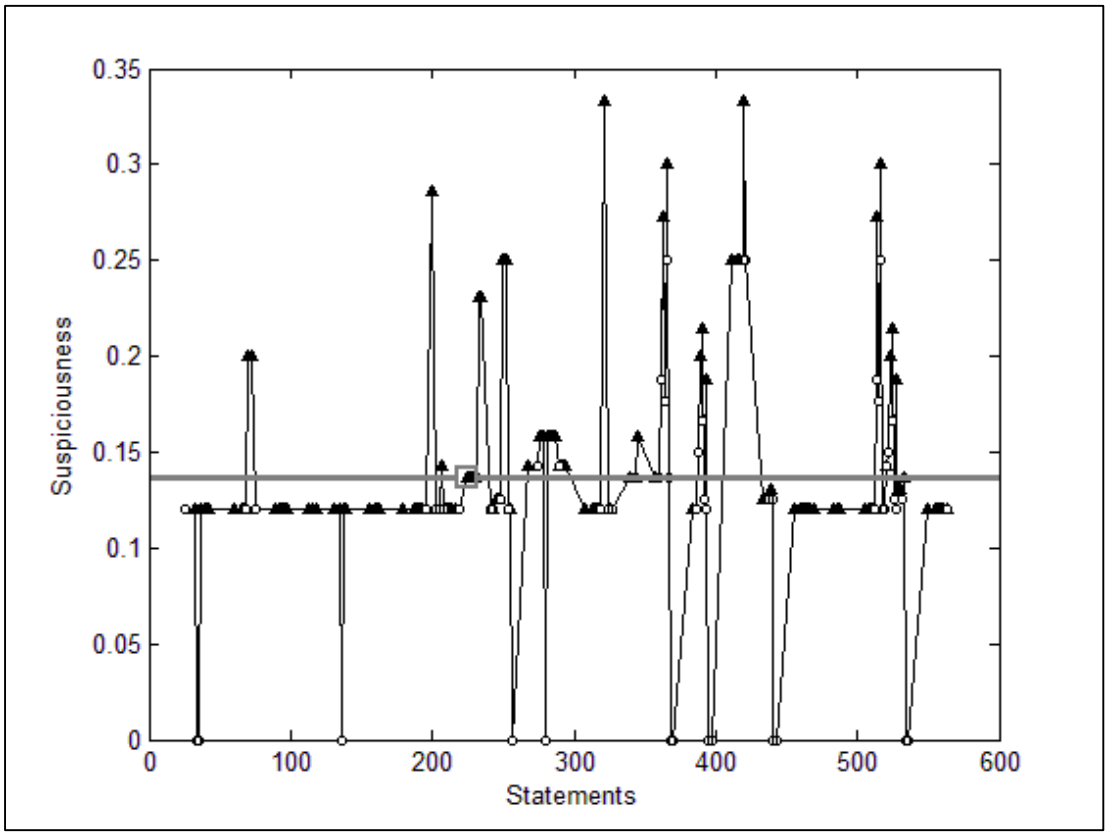

Figure 24 – Application of Local Maxima for Jaccard on PrintTokens_v01 with Distinct Test Suite

In Figure 24, there are 19 white circles which have higher suspiciousness values than faulty line must be excluded. Therefore, the rank of faulty line is decreased to 52 from 71.

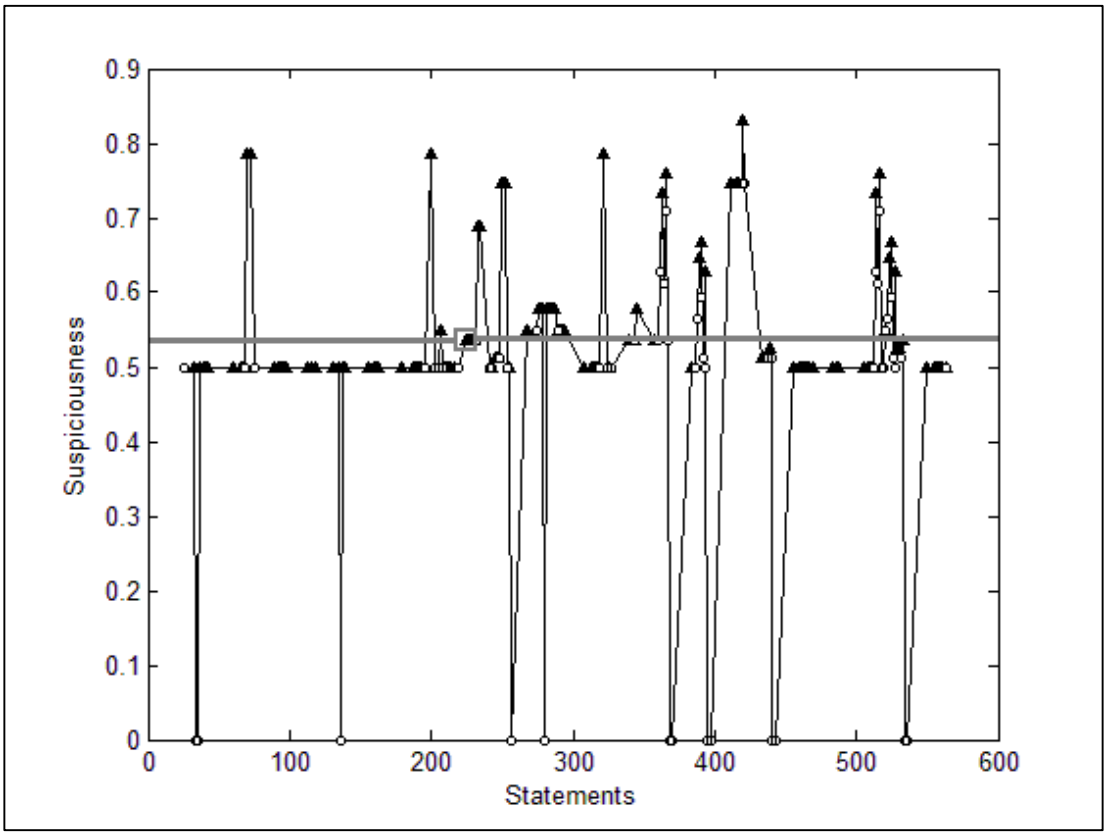

Figure 25 – Application of Local Maxima for Tarantula on PrintTokens_v01 with Distinct Test Suite

In Figure 25, there are 19 white circles which have higher suspiciousness values than faulty line must be excluded. Therefore, the rank of faulty line is decreased to 52 from 71.

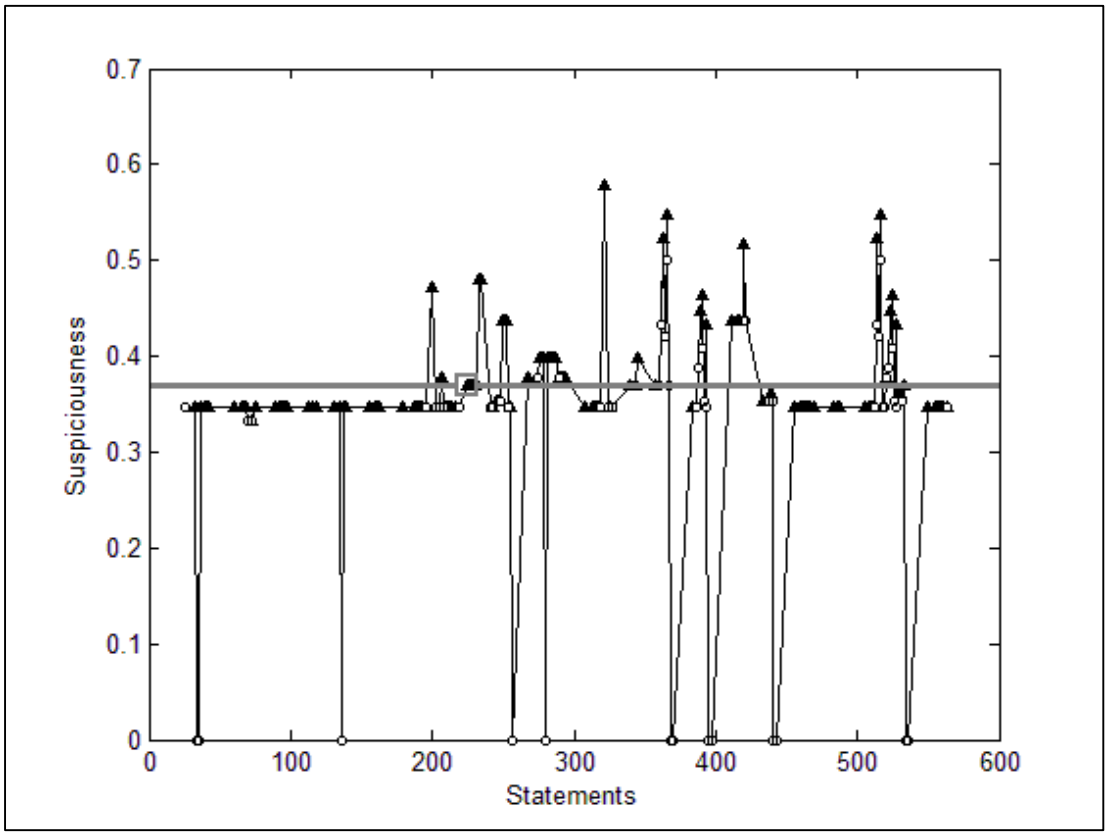

Figure 26 – Application of Local Maxima for Ochiai on PrintTokens_v01 with Distinct Test Suite

In Figure 26, there are 19 white circles which have higher suspiciousness values than faulty line must be excluded. Therefore, the rank of faulty line is decreased to 49 from 68.

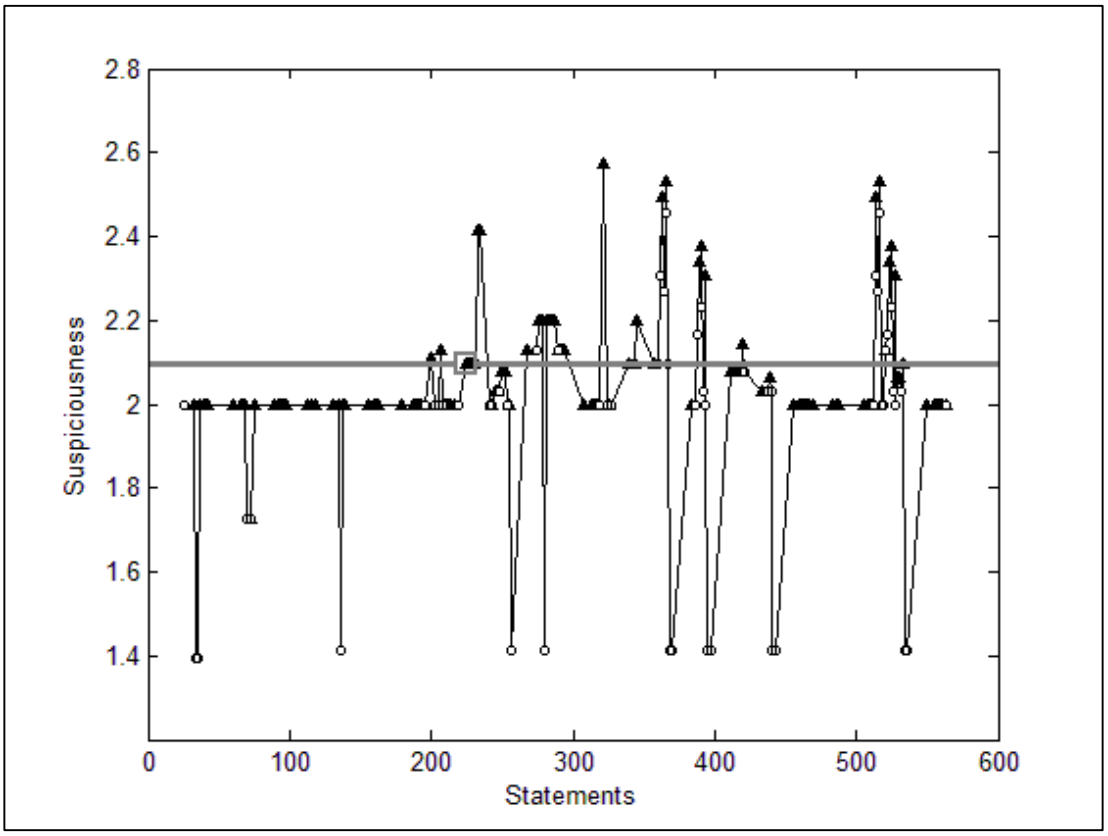

Figure 27 – Application of Local Maxima for Ela on PrintTokens_v01 with Distinct Test Suite

In Figure 27, there are 17 white circles which have higher suspiciousness values than faulty line must be excluded. Therefore, the rank of faulty line is decreased to 43 from 60.

There are four cases to be considered about the location of a statement for Local Maxima strategy since the process of finding the Local Maxima is different in these cases. First case occurs when the statement is the first statement of the program. Second case occurs when the statement is the last statement of the program. Third case occurs when the statement is the first statement of a method in the program. Fourth case occurs when the statement is the last statement of a method in the program. Border problems can arise in these cases. In order to handle these problems, we search for the local maxima around the only 1-right neighborhood of a statement for the cases 1 and 3 while we search for the local maxima around only the 1-left neighborhood of a statement for the cases 2 and 4.

6.3. Effect of LM on PrintTokens program

Table 13 shows the improvements of Local Maxima strategy on Ela and three prominent techniques on PrintTokens_v01 with Distinct Test Suite. The details of the results achieved

with Local Maxima for Redundant, Distinct, Distinct FminCov, Distinct FminCov Cluster, and Distinct FminCov Classify Test Suites are given in the section 7.7.

Table 13 – Improvement of Local Maxima Strategy for Expenses on PrintTokens_v01 with Distinct Test Suite

	Jaccard	Tarantula	Ochiai	Ela
Distinct Test Suite	21.45	21.45	20.54	18.13
Distinct Test Suite with Local Maxima Strategy	15.71	15.71	14.8	12.99
Improvement (%)	26.76	26.76	27.94	28.33

Table 13 indicates that Ela technique has a 28.33% improvement while Jaccard, Tarantula, and Ochiai techniques have 26.76%, 26.76%, and 27.94% improvements respectively.

CHAPTER 7

EXPERIMENTAL EVALUATION

In this chapter, we first introduce the subject programs that are used in our experiments. Next, we give the experimental results of the proposed fault localization technique (Ela) and compare it with three prominent fault localization techniques. Afterwards, we give the experimental result of the proposed test suite reduction technique and its improvements on the Ela and the three prominent fault localization techniques. Finally, we give the experimental result of the proposed Local Maxima technique and its improvements on the Ela and the three prominent fault localization techniques.

7.1. Subject Programs

In this study, we used the Siemens suite available at the software-artifact infrastructure repository as subject programs. We selected this suite since it is a quite frequently used benchmark suite in the fault localization and test reduction research. Siemens suite consists of seven C programs and associated test suites. For each of these programs, there are several versions, each of which contains manually injected one logical fault. There are 132 versions of C programs in this suite. For some of these versions, the test suite provided cannot differentiate the faulty version from the original program i.e. there were no failed tests for these versions. Since statistical fault localization techniques require at least one failed test, we excluded these versions in our experiments. We have used 118 versions of C programs as the subject programs in our study. We have added line breaks into the programs so that there would be one statement at each line. For the experiments, we needed to create a result vector and a coverage matrix for each version. To create the coverage matrix, we used gcov to collect the statement coverage. The repository contains an original version of the program as well as its mutants. We compiled all versions with gcc, run them with the tests, and record their outputs as text files. To determine whether a test has passed or failed on a mutant, we compared its output with the original version of the program. If both of them produce the

same output on the same test, we mark it as passed otherwise as failed. Table 14 gives the following information about these seven C programs: program name, line of code, number of all tests, versions of tests excluded, and a brief description of the program.

Table 14 – The seven C programs in SIR

Program	# of Versions	LOC	# of All Tests	Versions Excluded	Description
PrintTokens	7	565	4,130	V_{04},V_{06}	lexical analyzer
PrintTokens2	10	529	4,115	No	lexical analyzer
Replace	32	563	5,501	No	pattern replacement
Schedule	9	412	2,650	V_{09}	priority scheduler
Schedule2	10	307	2,588	V_{04},V_{09}	priority scheduler
Tcas	41	173	1,608	V_{13},V_{14},V_{15},V_{36},V_{38}	collision avoidance system
TotInfo	23	406	1,051	V_{06},V_{10},V_{19},V_{21}	information measurer

As discussed in Chapter 5, the existence of the equivalent tests can cause a bias in the ranking of statements. Equivalents of good tests increase the effectiveness and bad tests decrease the effectiveness of fault localization. Because whether a test is good test or not is not known, we select the safe side and eliminate the equivalent tests from both passed and failed tests in the experiments. The average number of tests used after the equivalent test elimination are given in Table 15.

Table 15 – The seven C programs in SIR after equivalent test elimination

Program	# of Versions	LOC	Average # of Tests Used	Versions Excluded	Description
PrintTokens	7	565	26	V_{04},V_{06}	lexical analyzer
PrintTokens2	10	529	25	No	lexical analyzer
Replace	32	563	23	No	pattern replacement
Schedule	9	412	9	V_{09}	priority scheduler
Schedule2	10	307	9	V_{04},V_{09}	priority scheduler
Tcas	41	173	6	V_{13},V_{14},V_{15},V_{36},V_{38}	collision avoidance system
TotInfo	23	406	9	V_{06},V_{10},V_{19},V_{21}	information measurer

7.2. Experimental Results I: Comparison with the Three Prominent Fault Localization Techniques on the Original Test Suite

In this section, we present the effectiveness of Ela in terms of expense and MRR in the experiments and compare it with those of the three prominent fault localization techniques which are Jaccard, Tarantula, and Ochiai on the original test suite. Recall that the original

suite contains many equivalent test cases. In the rest of this section, we call the experiments performed with the original test suite as "Redundant Test Suite".

Figure 28 shows a comparison of the four techniques on the seven subject programs with Redundant Test Suite. Each version in these seven programs has one fault. We need to examine 7.19% of the source code when the suspiciousness rankings computed by Ela until we find the faulty statement while 7.86%, 7.86%, 7.19% of the source code for the other three techniques: Jaccard, Tarantula, and Ochiai respectively for the PrintTokens program.

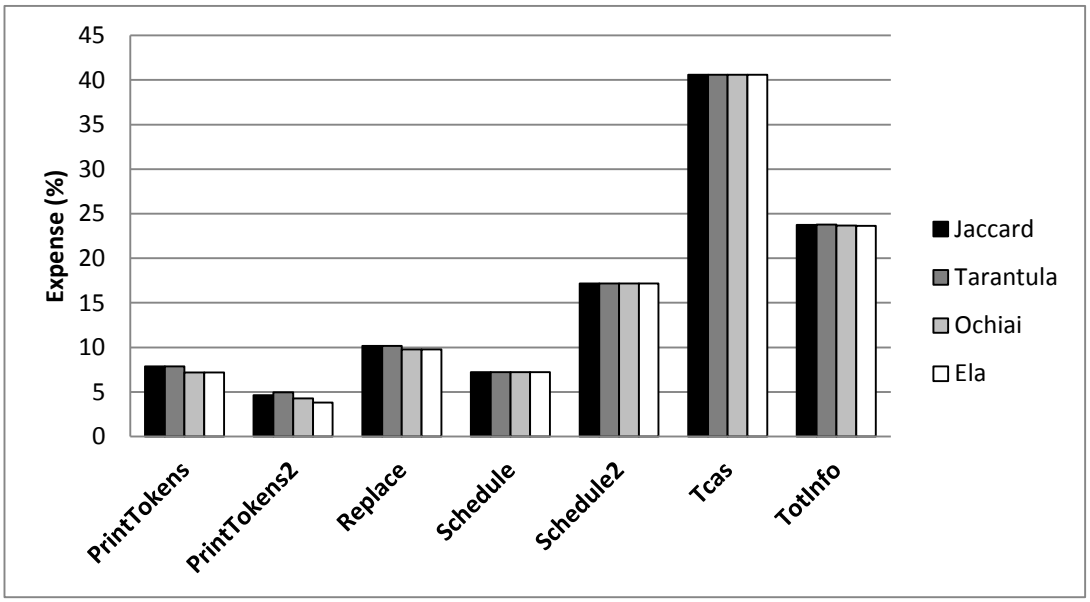

Figure 28 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Redundant Test Suite

From Figure 28, it is clear that Ela technique is superior to three prominent techniques under the specific conditions of our experiments. In the worst case of these experiments, the user still has to inspect on average only 40.57% of the code to find the fault by using Ela technique. In the best case, on average only 3.81% needs to be inspected. Ela technique achieves improvements ranging from 0.09% to 11.43% on average per program over the Ochiai technique which is the second best technique. An important conclusion drawn from Figure 28 is that under the specific conditions of our experiments, Ela technique gives a better effectiveness: it always performs at least as good as the other techniques, with an average improvement of 1.65% over the second best technique (Ochiai).

Table 16 displays the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Redundant Test Suite. Our aim is to compare Ela technique with three prominent techniques according their averaged expenses on

seven subject programs with Redundant Test Suite. As Table 16 shows, Ela has the lowest expense for the two subject programs and one of the lowest expenses for the five subject programs.

Table 16 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Redundant Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	7.86	7.86	7.19	7.19
PrintTokens2	4.65	4.97	4.3	3.81
Replace	10.16	10.16	9.77	9.77
Schedule	7.22	7.22	7.22	7.22
Schedule2	17.18	17.18	17.18	17.18
Tcas	40.57	40.57	40.57	40.57
TotInfo	23.74	23.78	23.66	23.63

Figure 29 shows a comparison of the four techniques on the seven subject programs with Redundant Test Suite in terms of their MRR values. Our purpose is to compare Ela technique with three prominent techniques across the versions of seven subject programs with Redundant Test Suite.

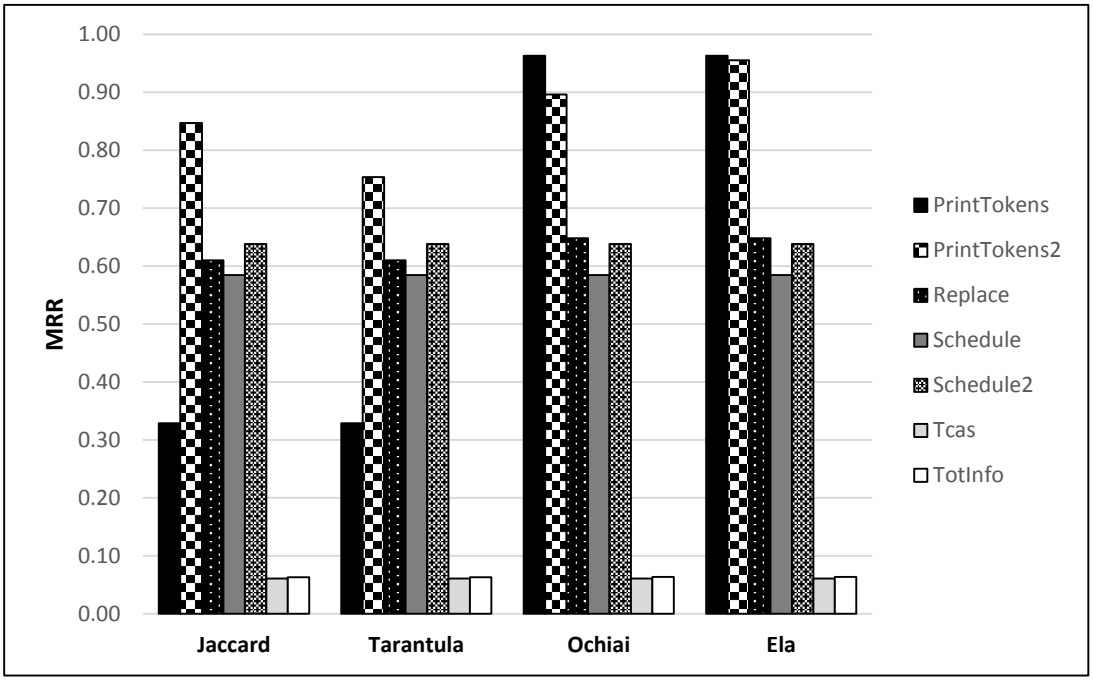

Figure 29 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Redundant Test Suite

Figure 29 indicates that Ela technique has the highest MRR value for the two subject programs and one of the highest MRR values for the five subject programs. Detailed

information about the MRR values of the four techniques with Redundant Test Suite on the seven programs is given in Table 17.

Table 17 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Redundant Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	0.3288	0.3288	0.9629	0.9629
PrintTokens2	0.8472	0.7536	0.8964	0.9551
Replace	0.6098	0.6098	0.6481	0.6481
Schedule	0.5846	0.5846	0.5846	0.5846
Schedule2	0.6381	0.6381	0.6381	0.6381
Tcas	0.0608	0.0608	0.0608	0.0608
TotInfo	0.0632	0.0632	0.0633	0.0634

The effectiveness of Ela and the three prominent fault localization techniques are compared in Table 18 in terms of their code examination efforts. The percentages of the code examined until the fault is found for each of the subject programs are given for Ela and the three prominent techniques. Table 18 indicates that Ela finds 35.6% of the total software faults with only 5% code examination effort while Jaccard, Tarantula, and Ochiai techniques find 33.6%, 33.9%, 35.6% of the total software faults for covering only 5% of the total source code on average of all the subject programs respectively.

Table 18 – Code examination efforts of four techniques on the subject programs

Code Examination Effort	Proportion of faults located			
	Jaccard	Tarantula	Ochiai	Ela
5	33.6%	33.9%	35.6%	35.6%
10	44.5%	44.9%	44.9%	44.9%
15	50.4%	50.8%	50.8%	50.8%
20	57.1%	57.6%	57.6%	57.6%
25	58%	57.6%	57.6%	57.6%
30	65.5%	65.3%	65.3%	65.3%
35	68.9%	68.6%	68.6%	68.6%
40	69.7%	69.5%	69.5%	69.5%
45	74.8%	74.6%	74.6%	74.6%
50+	100%	100%	100%	100%

Due to the large number of equivalent passed tests in Redundant Test Suite, the percentage of the failed tests is less than 5% for PrintTokens (1.31%), Replace (1.9%), Schedule (3.61%), Schedule2 (1.36%), and Tcas (2.31%). Therefore, it is difficult to differentiate the faulty statement from the innocent statements for these five test suites. On the other hand, the percentage of the failed tests is greater than 5% for PrintTokens2 (5.44%) and TotInfo

(8.34%) in Redundant Test Suite. Therefore, it is relatively easy to differentiate the faulty statement from the innocent statements. As a result, Ela technique differentiated the faulty statement from the innocent statements and achieved better fault localization effectiveness than the three prominent techniques for PrintTokens2 and TotInfo. Besides, it achieved same fault localization effectiveness for the remaining five programs.

7.2.1. Significance Analysis of Ela Effectiveness on Redundant Test Suite

We test whether Ela has statistically significant improvement over Jaccard, Tarantula, and Ochiai with Wilcoxon signed rank test at α=0.05 significance level for seven programs on Redundant Test Suite. The null hypothesis H_0 and the alternative hypothesis H_1 are as follows:

H_0: Median Expense of Ela is equal to Median Expense of Jaccard (Tarantula and Ochiai for other comparisons) on Redundant Test Suite.

H_1: Median Expense of Ela is less than Median Expense of Jaccard (Tarantula and Ochiai for other comparisons) on Redundant Test Suite.

Ela has statistically significant improvement over Jaccard for PrintTokens2 and Replace programs (p-value=0.016 and p-value=$6.1x10^{-5}$ respectively). Moreover, it has statistically significant improvement over Tarantula for PrintTokens2 and Replace programs (p-value=0.008 and p-value=$6.1x10^{-5}$ respectively). On the other hand, it does not have statistically significant improvement over Ochiai. The details of the significance analyses are given in the Appendix A.

To sum up, Ela technique has a higher ranking than three prominent techniques in 4 of 118 of all the versions of the subject programs while it is one of the best performing techniques for the remaining 114 versions. We can conclude that Ela technique is superior to the three prominent techniques on Redundant Test Suite.

7.3. Experiment II: Test Reduction Strategy I – Distinct Test Suite

In this experiment, we eliminated the equivalent tests and use the coverage matrix for the resulting suite of distinct tests. Using this input, we compare Ela with the three widely used SFL techniques. In the rest of this section, we refer the elimination of equivalent tests as the ‘Distinct Test Suite Strategy’.

7.3.1. Experimental Results II: Comparison with the Three Prominent Fault Localization Techniques with Distinct Test Suite

In this section, we present the effectiveness of Ela in terms of expense and MRR in the experiments and compare it with those of the three prominent fault localization techniques which are Jaccard, Tarantula, and Ochiai on Distinct Test Suite.

Figure 30 shows a comparison of the four techniques on the seven subject programs with Distinct Test Suite. Each version in these seven programs has one fault. We need to examine 6.59% of the source code when the suspiciousness rankings computed by Ela until we find the faulty statement while 7.55%, 8.4%, 7.19% of the source code for the other three techniques: Jaccard, Tarantula, and Ochiai respectively for the PrintTokens program.

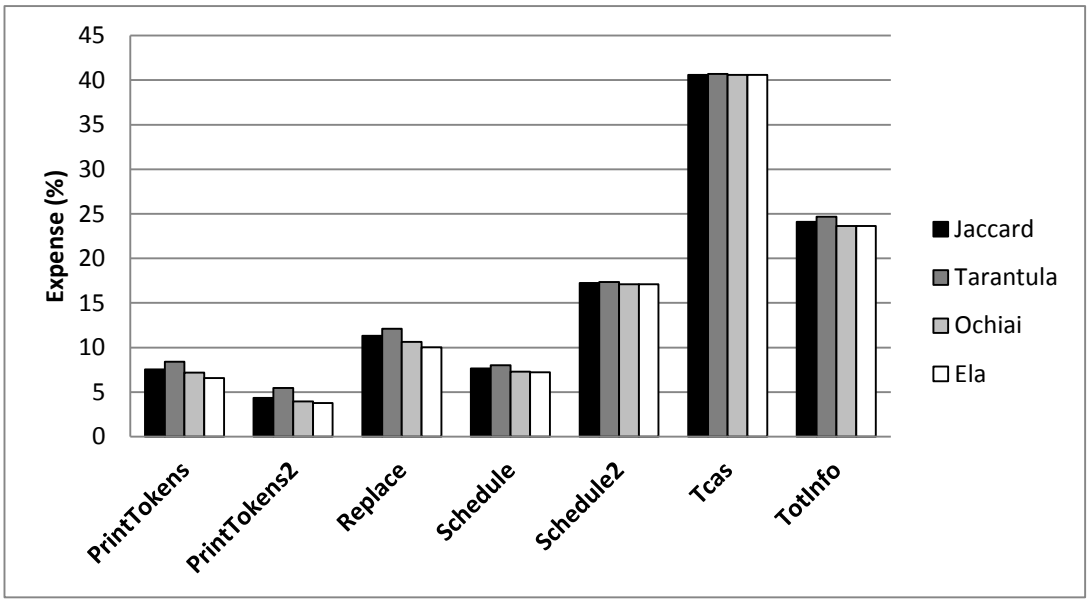

Figure 30 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct Test Suite

From Figure 30, it is clear that Ela technique is superior to three prominent techniques under the specific conditions of our experiments. In the worst case of these experiments, the user still has to inspect on average only 40.57% of the code to find the fault by using Ela technique. In the best case, on average only 3.78% needs to be inspected. Ela technique achieves improvements ranging from 1.2% to 8.4% on average per program over the Ochiai technique which is the second best technique. An important conclusion drawn from Figure 30 is that under the specific conditions of our experiments, Ela technique gives a better effectiveness: it always performs at least as good as the other techniques, with an average improvement of 2.81% over the second best technique (Ochiai).

Table 19 displays the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct Test Suite. Our aim is to compare Ela technique with three prominent techniques according their averaged expenses on seven subject programs with Distinct Test Suite. As Table 19 shows, Ela has the lowest expense for the four subject programs and one of the lowest expenses for the three subject programs.

Table 19 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	7.55	8.4	7.19	6.59
PrintTokens2	4.35	5.46	3.94	3.78
Replace	11.33	12.09	10.65	10.02
Schedule	7.65	8.01	7.3	7.22
Schedule2	17.23	17.37	17.09	17.09
Tcas	40.57	40.71	40.57	40.57
TotInfo	24.09	24.67	23.63	23.63

Figure 31 shows a comparison of the four techniques on the seven subject programs with Distinct Test Suite in terms of their MRR values. Our purpose is to compare Ela technique with three prominent techniques across the versions of seven subject programs with Distinct Test Suite.

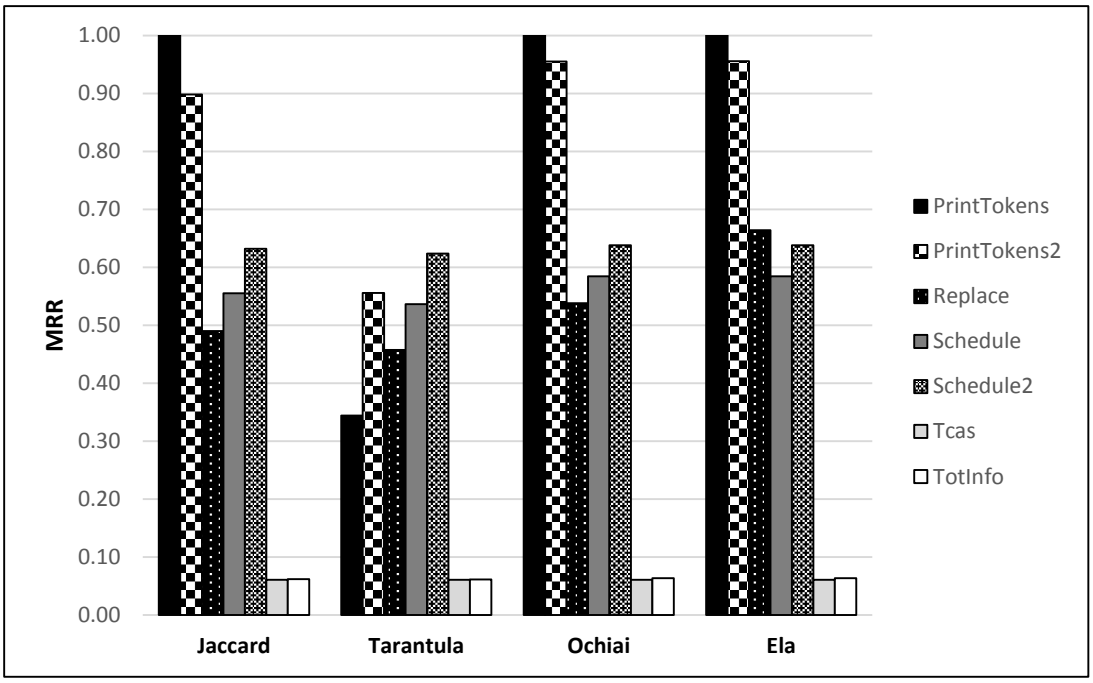

Figure 31 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct Test Suite

Figure 31 indicates that Ela technique has the highest MRR value for the four subject programs and one of the highest MRR values for the three subject programs. Detailed information about the MRR values of the four techniques with Distinct Test Suite on the seven programs is given in Table 20.

Table 20 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	1.4030	0.3442	1.4093	1.4116
PrintTokens2	0.8985	0.5556	0.9552	0.9558
Replace	0.4901	0.4575	0.5381	0.6638
Schedule	0.5553	0.5361	0.5845	0.5846
Schedule2	0.6320	0.6235	0.6381	0.6381
Tcas	0.0608	0.0607	0.0608	0.0608
TotInfo	0.0619	0.0612	0.0634	0.0634

The accuracies of Ela and the three prominent fault localization techniques are compared in Table 21 in terms of their code examination efforts. The percentages of the code examined until the fault is found for each of the subject programs are given for Ela and the three prominent techniques. Table 21 indicates that Ela finds 34.7% of the total software faults with only 5% code examination effort while Jaccard, Tarantula, and Ochiai techniques find 31.9%, 31.4%, 33.9% of the total software faults for covering only 5% of the total source code on average of all the subject programs.

Table 21 – Code examination efforts of four techniques on the subject programs

Code Examination Effort	Proportion of faults located			
	Jaccard	Tarantula	Ochiai	Ela
5	31.9%	31.4%	33.9%	34.7%
10	42%	40.7%	44.9%	44.9%
15	50.4%	50.8%	50.8%	50.8%
20	56.3%	56.8%	56.8%	57.6%
25	58%	57.6%	57.6%	57.6%
30	65.5%	65.3%	65.3%	65.3%
35	66.4%	66.1%	66.9%	68.6%
40	68.9%	67.8%	69.5%	69.5%
45	73.9%	70.3%	74.6%	74.6%
50+	100%	100%	100%	100%

After eliminating the equivalent tests from Redundant Test Suite, the percentage of the failed tests has been relatively increased in Distinct Test Suite. The percentage of the failed tests is greater than 5% for PrintTokens (15.01%), PrintTokens2 (11.74%), Replace (15.1%),

Schedule (18.02%), Schedule2 (16.74%), Tcas (20.12%), and TotInfo (31.41%). Therefore, it is relatively easy to differentiate the faulty statement from the innocent statements in Distinct Test Suite. However, there are some relatively difficult types of faulty statements to be located in PrintTokens (Comment or Delete method call; Comment or Delete variable assignment), PrintTokens2 (Comment or Delete return value; Change variable assignment), Replace (Change method call; Change method body; Change variable assignment; Change or Comment or Delete variable assignment; Change variable initialization; Change lines of code), and Schedule (Change variable assignment; Change or Comment or Delete variable assignment) programs. The list of fault types are given in Table 56 in the Appendix B. Consequently, it is relatively difficult to locate the faulty statements for these version of the programs in the Siemens test suite.

7.3.2. Significance Analysis of Ela Effectiveness on Distinct Test Suite

We test whether Ela has statistically significant improvement over Jaccard, Tarantula, and Ochiai or not with Wilcoxon signed rank test at α=0.05 significance level for seven programs on Distinct Test Suite. The null hypothesis H_0 and the alternative hypothesis H_1 are as follows:

H_0: Median Expense of Ela is equal to Median Expense of Jaccard (Tarantula and Ochiai for other comparisons) on Distinct Test Suite.

H_1: Median Expense of Ela is less than Median Expense of Jaccard (Tarantula and Ochiai for other comparisons) on Distinct Test Suite.

Ela has statistically significant improvement over Jaccard for PrintTokens2, Replace, and TotInfo programs (p-value=0.031, p-value=$7.5x10^{-9}$, and p-value=$9.8x10^{-4}$ respectively). Moreover, it has statistically significant improvement over Tarantula for PrintTokens, PrintTokens2, Replace, Tcas, and TotInfo programs (p-value=0.031, p-value=0.002, p-value=$7.5x10^{-9}$, p-value=0.016, and p-value=$2.4x10^{-4}$ respectively). Furthermore, it has statistically significant improvement over Ochiai for Replace program (p-value=$7.5x10^{-9}$). The details of the significance analyses are given in the Appendix A.

To sum up, Ela technique has a higher ranking than three prominent techniques in 31 of 118 of all the versions of the subject programs while it is one of the best performing techniques for the remaining 87 versions. We can conclude that Ela technique is superior to the three prominent techniques on Distinct Test Suite.

7.4. Experiment III: Test Reduction Strategy II – Distinct FminCov Test Suite

We applied the failed test reduction strategy on four SFL techniques. In this experiment, we eliminated the equivalent tests and use the coverage matrix for the resulting suite of distinct tests to eliminate the bias.

7.4.1. Experimental Results III: Comparison with the Three Prominent Fault Localization Techniques with Distinct FminCov Test Suite

In this section, we present the effectiveness of Ela in terms of expense and MRR in the experiments and compare it with those of the three prominent fault localization techniques which are Jaccard, Tarantula, and Ochiai on Distinct FminCov Test Suite. On average 10.1% test size reduction is achieved with this strategy.

Figure 32 shows a comparison of the four techniques on the seven subject programs with Distinct FminCov Test Suite. Each version in these seven programs has one fault. We need to examine 6.47% of the source code until we find the faulty statement for the PrintTokens program with all of the four techniques: Ela, Jaccard, Tarantula, and Ochiai.

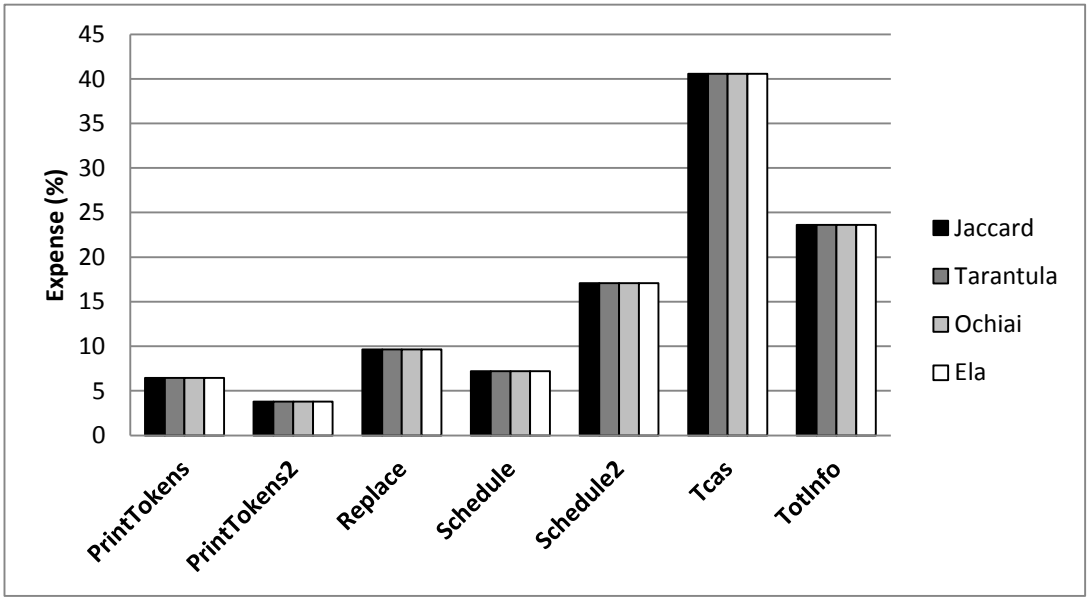

Figure 32 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Test Suite

From Figure 32, it is clear that Ela technique is superior to three prominent techniques under the specific conditions of our experiments. In the worst case of these experiments, the user still has to inspect on average only 40.57% of the code to find the fault by using Ela technique. In the best case, on average only 3.78% needs to be inspected. Ela technique achieves same performance with other three techniques.

Table 22 displays the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Test Suite. Our aim is to compare Ela technique with three prominent techniques according their averaged expenses on seven subject programs with Distinct FminCov Test Suite. As Table 22 shows, Ela has one of the lowest expenses for each of the programs.

Table 22 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	6.47	6.47	6.47	6.47
PrintTokens2	3.78	3.78	3.78	3.78
Replace	9.63	9.63	9.63	9.63
Schedule	7.22	7.22	7.22	7.22
Schedule2	17.09	17.09	17.09	17.09
Tcas	40.57	40.57	40.57	40.57
TotInfo	23.61	23.61	23.61	23.61

Figure 33 shows a comparison of the four techniques on the seven subject programs with Distinct FminCov Test Suite in terms of their MRR values. Our purpose is to compare Ela technique with three prominent techniques across the versions of seven subject programs with Distinct FminCov Test Suite.

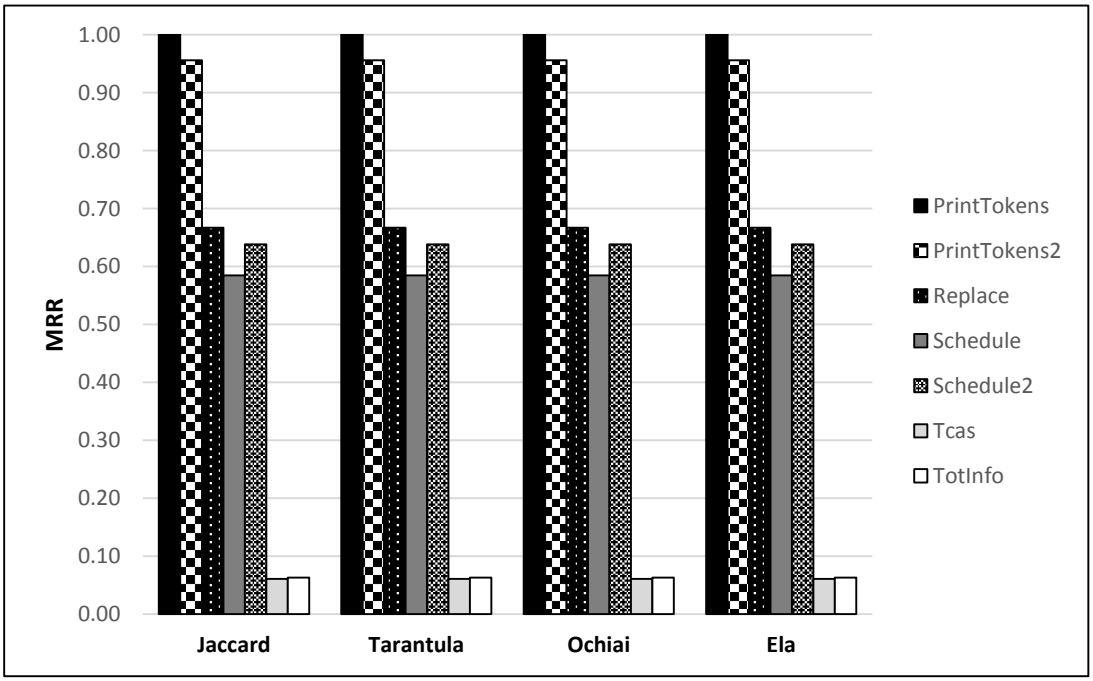

Figure 33 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Test Suite

Figure 33 indicates that Ela technique has one of the highest MRR values for each of the seven subject programs. Detailed information about the MRR values of the four techniques with Distinct FminCov Test Suite on the seven programs is given in Table 23.

Table 23 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	1.4120	1.4120	1.4120	1.4120
PrintTokens2	0.9558	0.9558	0.9558	0.9558
Replace	0.6667	0.6667	0.6667	0.6667
Schedule	0.5846	0.5846	0.5846	0.5846
Schedule2	0.6381	0.6381	0.6381	0.6381
Tcas	0.0608	0.0608	0.0608	0.0608
TotInfo	0.0634	0.0634	0.0634	0.0634

The accuracies of Ela and the three prominent fault localization techniques are compared in Table 24 in terms of their code examination efforts. The percentages of the code examined until the fault is found for each of the subject programs are given for Ela and the three prominent techniques. Table 24 indicates that Ela, Jaccard, Tarantula, and Ochiai techniques find 35.6% of the total software faults with only 5% code examination effort, i.e. covering only 5% of the total source code on average of all the subject programs.

Table 24 – Code examination efforts of four techniques on the subject programs

Code Examination Effort	Proportion of faults located			
	Jaccard	Tarantula	Ochiai	Ela
5	35.6%	35.6%	35.6%	35.6%
10	44.9%	44.9%	44.9%	44.9%
15	50.8%	50.8%	50.8%	50.8%
20	57.6%	57.6%	57.6%	57.6%
25	57.6%	57.6%	57.6%	57.6%
30	65.3%	65.3%	65.3%	65.3%
35	68.6%	68.6%	68.6%	68.6%
40	69.5%	69.5%	69.5%	69.5%
45	74.6%	74.6%	74.6%	74.6%
50+	100%	100%	100%	100%

For each of the subject programs in Distinct FminCov Test Suite, there is only one failed test which is not subsumed by any other failed test (called FminCov failed test). Thus, the percentage of the failed tests is very small compared to Redundant and Distinct test suites. Consequently, it is difficult to differentiate the faulty statement from the innocent statements for this test suite. We can say that if we had a test suite with more than one FminCov failed

test, the faulty statement would be differentiated from the innocent statements and Ela would achieve better fault localization effectiveness than the three prominent fault localization techniques for this test suite.

7.4.2. Significance Analysis of Ela Effectiveness on Distinct FminCov Test Suite

We test whether Ela has statistically significant improvement over Jaccard, Tarantula, and Ochiai or not with Wilcoxon signed rank test at α=0.05 significance level for seven programs on Distinct FminCov Test Suite. The null hypothesis H_0 and the alternative hypothesis H_1 are as follows:

H_0: Median Expense of Ela is equal to Median Expense of Jaccard (Tarantula and Ochiai for other comparisons) on Distinct FminCov Test Suite.

H_1: Median Expense of Ela is less than Median Expense of Jaccard (Tarantula and Ochiai for other comparisons) on Distinct FminCov Test Suite.

Ela does not have statistically significant improvement over Jaccard, Tarantula and Ochiai. The details of the significance analyses are given in the Appendix A.

7.4.3. Significance Analysis of Failed Test Reduction

We test whether there is a statistically significant failed test reduction or not with Wilcoxon signed rank test at α=0.05 significance level. The null hypothesis H_0 and the alternative hypothesis H_1 are as follows:

H_0: Median of failed test reduction is zero.
H_1: Median of failed test reduction is greater than zero.

There is a statistically significant failed test reduction (p-value=0.0156).

7.5. Experiment IV: Test Reduction Strategy III - Distinct FminCov Cluster Test Suite

We applied the clustering based passed test reduction strategy on four SFL techniques. In this experiment, we used the coverage matrix of the test suite after the elimination of the equivalent tests to remove the bias.

7.5.1. Experimental Results IV: Comparison with the Three Prominent Fault Localization Techniques with Distinct FminCov Cluster Test Suite

In this section, we present the effectiveness of Ela in terms of expense and MRR in the experiments and compare it with those of the three prominent fault localization techniques which are Jaccard, Tarantula, and Ochiai on Distinct FminCov Cluster Test Suite. On average 34.1% test size reduction is achieved with this strategy.

Figure 34 shows a comparison of the four techniques on the seven subject programs with Distinct FminCov Cluster Test Suite. Each version in these seven programs has one fault. We need to examine 13.29% of the source code until we find the faulty statement for the PrintTokens program with all of the four techniques: Ela, Jaccard, Tarantula, and Ochiai.

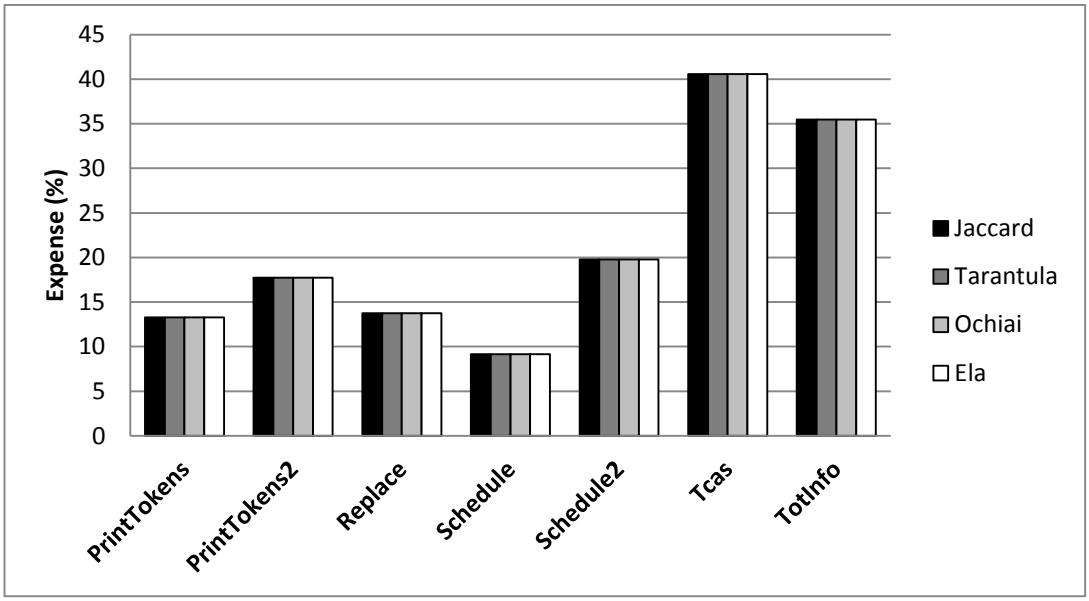

Figure 34 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Cluster Test Suite

From Figure 34, it is clear that Ela technique is superior to three prominent techniques under the specific conditions of our experiments. In the worst case of these experiments, the user still has to inspect on average only 40.57% of the code to find the fault by using Ela technique. In the best case, on average only 9.14% needs to be inspected. Ela technique achieves same performance with other three techniques.

Table 25 displays the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Cluster Test Suite. Our aim is to compare Ela technique with three prominent techniques according their averaged

expenses on seven subject programs with Distinct FminCov Cluster Test Suite. As Table 25 shows, Ela has one of the lowest expenses for each of the programs.

Table 25 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Cluster Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	13.29	13.29	13.29	13.29
PrintTokens2	17.73	17.73	17.73	17.73
Replace	13.75	13.75	13.75	13.75
Schedule	9.14	9.14	9.14	9.14
Schedule2	19.78	19.78	19.78	19.78
Tcas	40.57	40.57	40.57	40.57
TotInfo	35.48	35.48	35.48	35.48

Figure 35 shows a comparison of the four techniques on the seven subject programs with Distinct FminCov Cluster Test Suite in terms of their MRR values. Our purpose is to compare Ela technique with three prominent techniques across the versions of seven subject programs with Distinct FminCov Cluster Test Suite.

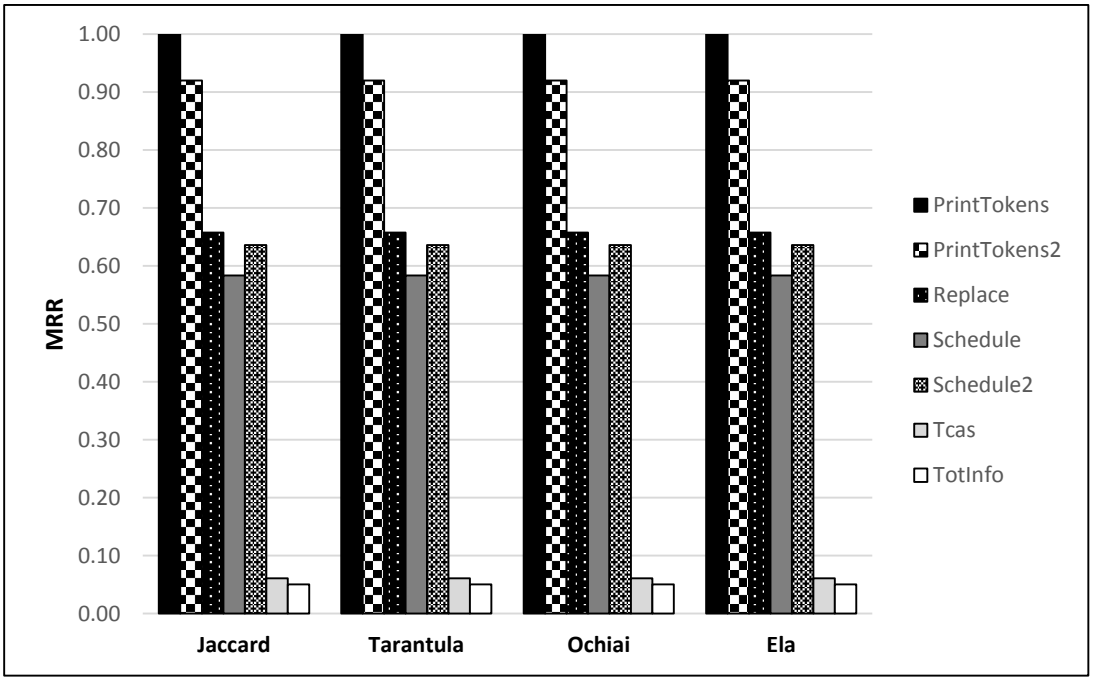

Figure 35 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Cluster Test Suite

Figure 35 indicates that Ela technique has one of the highest MRR values for each of the seven subject programs. Detailed information about the MRR values of the four techniques with Distinct FminCov Cluster Test Suite on the seven programs is given in Table 26.

Table 26 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Cluster Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	1.4044	1.4044	1.4044	1.4044
PrintTokens2	0.9198	0.9198	0.9198	0.9198
Replace	0.6574	0.6574	0.6574	0.6574
Schedule	0.5835	0.5835	0.5835	0.5835
Schedule2	0.6363	0.6363	0.6363	0.6363
Tcas	0.0608	0.0608	0.0608	0.0608
TotInfo	0.0505	0.0505	0.0505	0.0505

The accuracies of Ela and the three prominent fault localization techniques are compared in Table 27 in terms of their code examination efforts. The percentages of the code examined until the fault is found for each of the subject programs are given for Ela and the three prominent techniques. Table 27 indicates that Ela, Jaccard, Tarantula, and Ochiai techniques find 33.9% of the total software faults with only 5% code examination effort, i.e. covering only 5% of the total source code on average of all the subject programs.

Table 27 – Code examination efforts of four techniques on the subject programs

Code Examination Effort	Proportion of faults located			
	Jaccard	Tarantula	Ochiai	Ela
5	33.9%	33.9%	33.9%	33.9%
10	42.4%	42.4%	42.4%	42.4%
15	46.6%	46.6%	46.6%	46.6%
20	46.6%	46.6%	46.6%	46.6%
25	46.6%	46.6%	46.6%	46.6%
30	46.6%	46.6%	46.6%	46.6%
35	51.7%	51.7%	51.7%	51.7%
40	54.2%	54.2%	54.2%	54.2%
45	64.4%	64.4%	64.4%	64.4%
50	96.6%	96.6%	96.6%	96.6%
55	98.3%	98.3%	98.3%	98.3%
60+	100%	100%	100%	100%

Same reason in the subsection 7.4.1 is hold in Distinct FminCov Cluster Test Suite. Since the percentage of the failed tests is very small compared to Redundant and Distinct test suites, it is difficult to differentiate the faulty statement from the innocent statements for this test suite. We can say that if we had a test suite with more than one FminCov failed test, the faulty statement would be differentiated from the innocent statements and Ela would achieve better fault localization effectiveness the three prominent fault localization techniques for this test suite.

7.5.2. Significance Analysis of Ela Effectiveness on Distinct FminCov Cluster Test Suite

We test whether Ela has statistically significant improvement over Jaccard, Tarantula, and Ochiai or not with Wilcoxon signed rank test at α=0.05 significance level for seven programs on Distinct FminCov Cluster Test Suite. The null hypothesis H_0 and the alternative hypothesis H_1 are as follows:

H_0: Median Expense of Ela is equal to Median Expense of Jaccard (Tarantula and Ochiai for other comparisons) on Distinct FminCov Cluster Test Suite.

H_1: Median Expense of Ela is less than Median Expense of Jaccard (Tarantula and Ochiai for other comparisons) on Distinct FminCov Cluster Test Suite.

Ela does not have statistically significant improvement over Jaccard, Tarantula and Ochiai. The details of the significance analyses are given in the Appendix A.

7.5.3. Significance Analysis of Passed Test Reduction with Clustering

We test whether there is a statistically significant passed test reduction with clustering or not with Wilcoxon signed rank test at α=0.05 significance level. The null hypothesis H_0 and the alternative hypothesis H_1 are as follows:

H_0: Median of passed test reduction with clustering is zero.
H_1: Median of passed test reduction with clustering is greater than zero.

There is a statistically significant passed test reduction with clustering (p-value=0.0078).

7.6. Experiment V: Test Reduction Strategy IV - Distinct FminCov Classify Test Suite

We applied the classification based passed test reduction strategy in combination with FminCov Test Suite Strategy on four SFL techniques. In this experiment, we used the coverage matrix of the test suite after the elimination of the equivalent tests to remove the bias.

7.6.1. Experimental Results V: Comparison with the Three Prominent Fault Localization Techniques with Distinct FminCov Classify Test Suite

In this section, we present the effectiveness of Ela in terms of expense and MRR in the experiments and compare it with those of the three prominent fault localization techniques

which are Jaccard, Tarantula, and Ochiai on Distinct FminCov Classify Test Suite. On average 30.3% test size reduction is achieved with this strategy.

Figure 36 shows a comparison of the four techniques on the seven subject programs with Distinct FminCov Classify Test Suite. Each version in these seven programs has one fault. We need to examine 6.47% of the source code until we find the faulty statement for the PrintTokens program with all of the four techniques: Ela, Jaccard, Tarantula, and Ochiai.

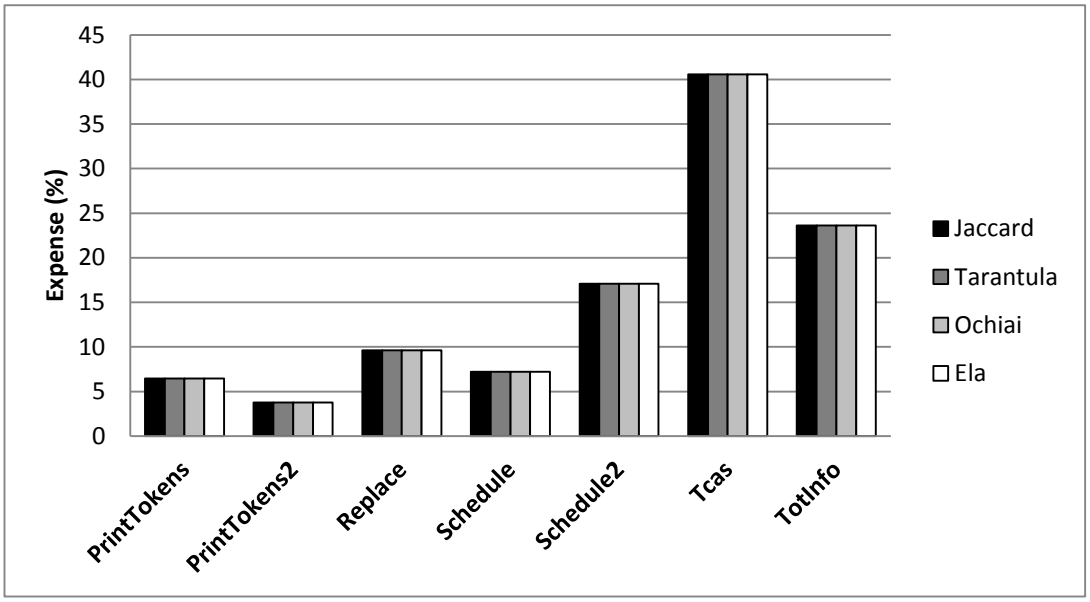

Figure 36 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Classify Test Suite

From Figure 36, it is clear that Ela technique is superior to three prominent techniques under the specific conditions of our experiments. In the worst case of these experiments, the user still has to inspect on average only 40.57% of the code to find the fault by using Ela technique. In the best case, on average only 3.78% needs to be inspected. Ela technique achieves same performance with other three techniques.

Table 28 displays the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Classify Test Suite. Our aim is to compare Ela technique with three prominent techniques according their averaged expenses on seven subject programs with Distinct FminCov Classify Test Suite. As Table 28 shows, Ela has one of the lowest expenses for each of the programs.

Table 28 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Classify Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	6.47	6.47	6.47	6.47
PrintTokens2	3.78	3.78	3.78	3.78
Replace	9.63	9.63	9.63	9.63
Schedule	7.22	7.22	7.22	7.22
Schedule2	17.09	17.09	17.09	17.09
Tcas	40.57	40.57	40.57	40.57
TotInfo	23.61	23.61	23.61	23.61

Figure 37 shows a comparison of the four techniques on the seven subject programs with Distinct FminCov Classify Test Suite in terms of their MRR values. Our purpose is to compare Ela technique with three prominent techniques across the versions of seven subject programs with Distinct FminCov Classify Test Suite.

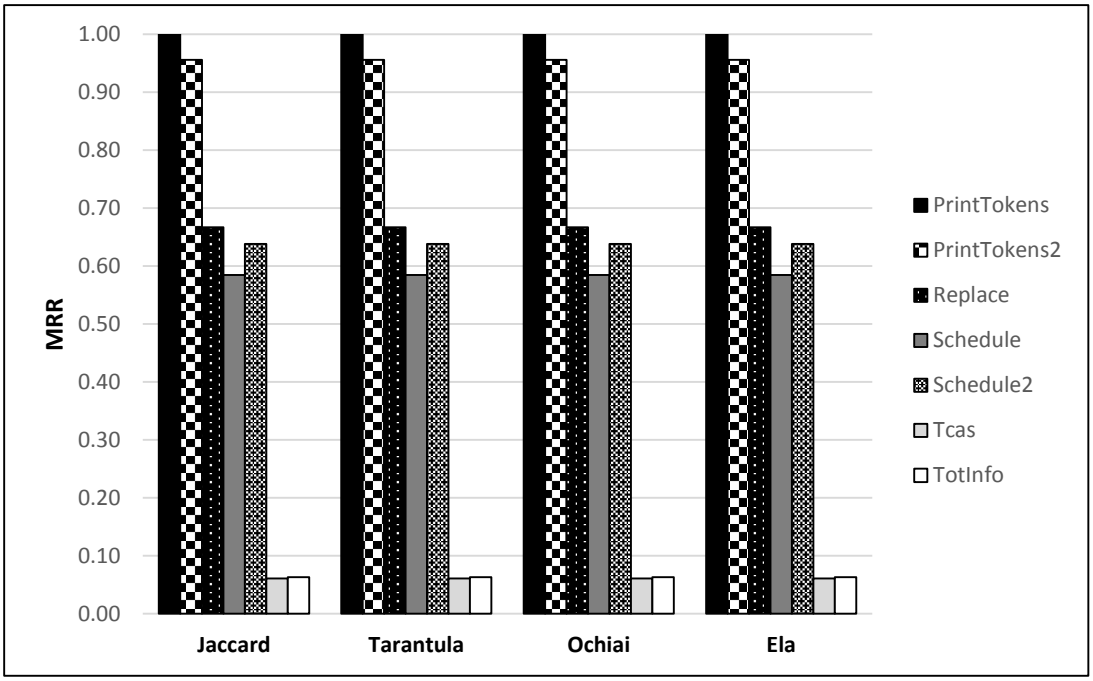

Figure 37 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Classify Test Suite

Figure 37 indicates that Ela technique has one of the highest MRR values for each of the seven subject programs. Detailed information about the MRR values of the four techniques with Distinct FminCov Classify Test Suite on the seven programs is given in Table 29.

Table 29 – MRR values for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with Distinct FminCov Classify Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	1.4120	1.4120	1.4120	1.4120
PrintTokens2	0.9558	0.9558	0.9558	0.9558
Replace	0.6667	0.6667	0.6667	0.6667
Schedule	0.5846	0.5846	0.5846	0.5846
Schedule2	0.6381	0.6381	0.6381	0.6381
Tcas	0.0608	0.0608	0.0608	0.0608
TotInfo	0.0634	0.0634	0.0634	0.0634

The accuracies of Ela and the three prominent fault localization techniques are compared in Table 30 in terms of their code examination efforts. The percentages of the code examined until the fault is found for each of the subject programs are given for Ela and the three prominent techniques. Table 30 indicates that Ela, Jaccard, Tarantula, and Ochiai techniques find 35.6% of the total software faults with only 5% code examination effort, i.e. covering only 5% of the total source code on average of all the subject programs.

Table 30 – Code examination efforts of four techniques on the subject programs

Code Examination Effort	Proportion of faults located			
	Jaccard	Tarantula	Ochiai	Ela
5	35.6%	35.6%	35.6%	35.6%
10	44.9%	44.9%	44.9%	44.9%
15	50.8%	50.8%	50.8%	50.8%
20	57.6%	57.6%	57.6%	57.6%
25	57.6%	57.6%	57.6%	57.6%
30	65.3%	65.3%	65.3%	65.3%
35	68.6%	68.6%	68.6%	68.6%
40	69.5%	69.5%	69.5%	69.5%
45	74.6%	74.6%	74.6%	74.6%
50+	100%	100%	100%	100%

Same reason in the subsection 7.4.1 is hold in Distinct FminCov Classify Test Suite. Since the percentage of the failed tests is very small compared to Redundant and Distinct test suites, it difficult enough to differentiate the faulty statement from the innocent statements for this test suite. We can say that if we had a test suite with more than one FminCov failed test, the faulty statement would be differentiated from the innocent statements and Ela would achieve better fault localization effectiveness the three prominent fault localization techniques for this test suite.

7.6.2. Significance Analysis of Ela Effectiveness on Distinct FminCov Classify Test Suite

We test whether Ela has statistically significant improvement over Jaccard, Tarantula, and Ochiai or not with Wilcoxon signed rank test at α=0.05 significance level for seven programs on Distinct FminCov Classify Test Suite. The null hypothesis H_0 and the alternative hypothesis H_1 are as follows:

H_0: Median Expense of Ela is equal to Median Expense of Jaccard (Tarantula and Ochiai for other comparisons) on Distinct FminCov Classify Test Suite.

H_1: Median Expense of Ela is less than Median Expense of Jaccard (Tarantula and Ochiai for other comparisons) on Distinct FminCov Classify Test Suite.

Ela does not have statistically significant improvement over Jaccard, Tarantula and Ochiai. The details of the significance analyses are given in the Appendix A.

7.6.3. Significance Analysis of Passed Test Reduction with Classification

We test whether there is a statistically significant passed test reduction with classification or not with Wilcoxon signed rank test at α=0.05 significance level. The null hypothesis H_0 and the alternative hypothesis H_1 are as follows:

H_0: Median of passed test reduction with classification is zero.
H_1: Median of passed test reduction with classification is greater than zero.

There is a statistically significant passed test reduction with classification (p-value=0.0078).

7.7. Experiment VI: Effective Ranking Strategy: Local Maxima

We applied one post processing strategy called local maxima strategy to rank statements effectively.

7.7.1. Effective Ranking Strategy on Redundant Test Suite

In this section, we present the effectiveness of Ela in terms of expense and MRR in the experiments and compare it with those of the three prominent fault localization techniques which are Jaccard, Tarantula, and Ochiai with LM strategy on Redundant Test Suite.

Figure 38 shows a comparison of the four techniques on the seven subject programs for LM strategy on Redundant Test Suite. Each version in these seven programs has one fault. We need to examine 5.44% of the source code when the suspiciousness rankings computed by Ela until we find the faulty statement while 6.1%, 6.1%, 5.44% of the source code for the other three techniques: Jaccard, Tarantula, and Ochiai respectively for the PrintTokens program.

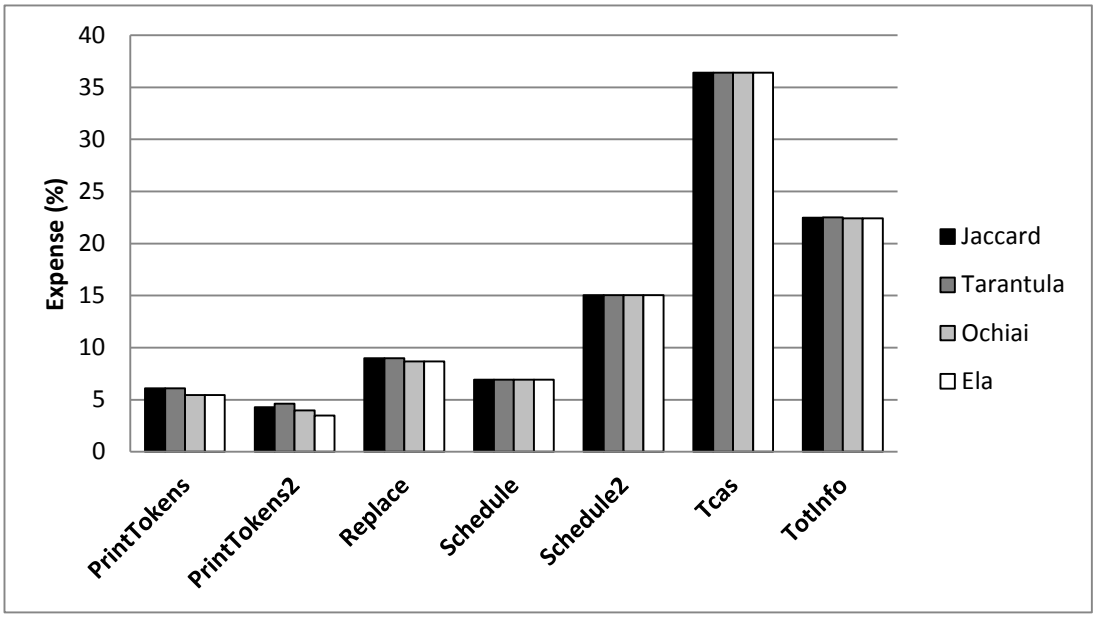

Figure 38 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Redundant Test Suite

Table 31 displays the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Redundant Test Suite. As Table 31 shows, Ela has the lowest expense for the two subject programs and one of the lowest expenses for the five subject programs.

Table 31 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Redundant Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	7.86	7.86	7.19	7.19
PrintTokens2	4.65	4.97	4.3	3.81
Replace	10.16	10.16	9.77	9.77
Schedule	7.22	7.22	7.22	7.22
Schedule2	17.18	17.18	17.18	17.18
Tcas	40.57	40.57	40.57	40.57
TotInfo	23.74	23.78	23.66	23.63

Table 32 shows the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Redundant Test Suite.

Table 32 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Redundant Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	6.1	6.1	5.44	5.44
PrintTokens2	4.3	4.62	3.97	3.48
Replace	8.99	8.99	8.69	8.69
Schedule	6.91	6.91	6.91	6.91
Schedule2	15.03	15.03	15.03	15.03
Tcas	36.42	36.42	36.42	36.42
TotInfo	22.47	22.52	22.43	22.41

From Table 32, it is clear that Ela technique is superior to three prominent techniques under the specific conditions of our experiments. In the worst case of these experiments, the user still has to inspect on average only 36.42% of the code to find the fault by using Ela technique. In the best case, on average only 3.48% needs to be inspected. Ela technique achieves improvements ranging from 0.1% to 12.37% on average per program over the Ochiai technique which is the second best technique.

Table 32 shows that under the specific conditions of our experiments, Ela technique gives a better effectiveness: it always performs at least as good as the other techniques, with an average improvement of 1.78% over the second best technique (Ochiai).

Figure 39 shows LM Improvement over averaged expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Redundant Test Suite.

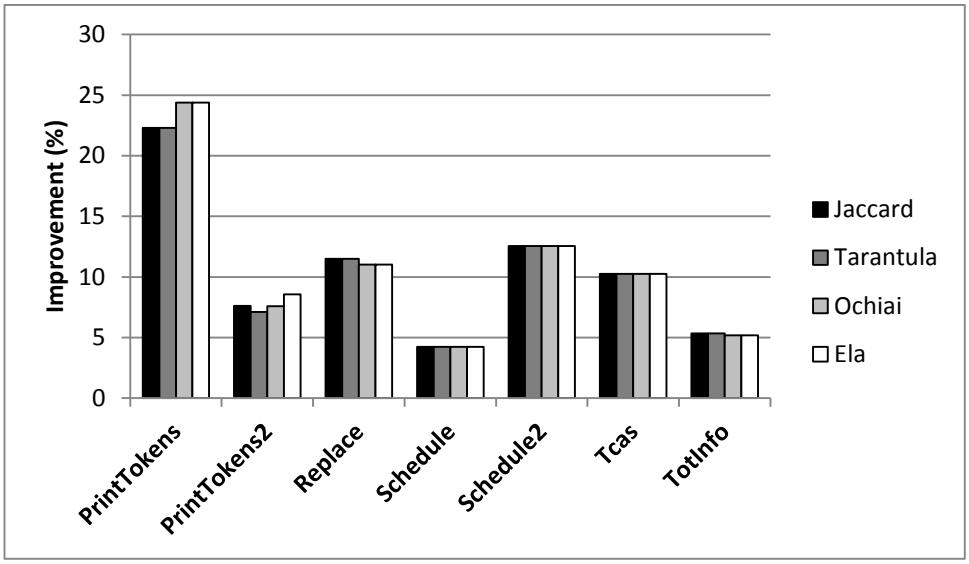

Figure 39 – LM Improvement over Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques for seven programs on Redundant Test Suite

Figure 39 indicates that Ela technique has 10.88% averaged improvement while Jaccard, Tarantula, and Ochiai techniques have 10.54%, 10.47%, and 10.74% averaged improvements respectively on average of all subject programs. There is approximately 10% averaged improvement of LM strategy for all techniques on Redundant Test Suite.

7.7.1.1. Significance Analysis of Local Maxima with Ela Technique on Redundant Test Suite

We test whether there is a statistically significant improvement of local maxima with Ela or not with Wilcoxon signed rank test at α=0.05 significance level. The null hypothesis H_0 and the alternative hypothesis H_1 are as follows:

H_0: Median Expense Improvement of Local Maxima is zero on Redundant Test Suite.
H_1: Median Expense Improvement of Local Maxima is greater than zero on Redundant Test Suite.

There is a statistically significant improvement of local maxima with Ela (p-value=0.0078) on Redundant Test Suite.

7.7.2. Effective Ranking Strategy on Distinct Test Suite

In this section, we present the effectiveness of Ela in terms of expense and MRR in the experiments and compare it with those of the three prominent fault localization techniques which are Jaccard, Tarantula, and Ochiai with LM strategy on Distinct Test Suite.

Figure 40 shows a comparison of the four techniques on the seven subject programs for LM strategy on Distinct Test Suite. Each version in these seven programs has one fault. We need to examine 5.08% of the source code when the suspiciousness rankings computed by Ela until we find the faulty statement while 5.86%, 6.53%, 5.56% of the source code for the other three techniques: Jaccard, Tarantula, and Ochiai respectively for the PrintTokens program.

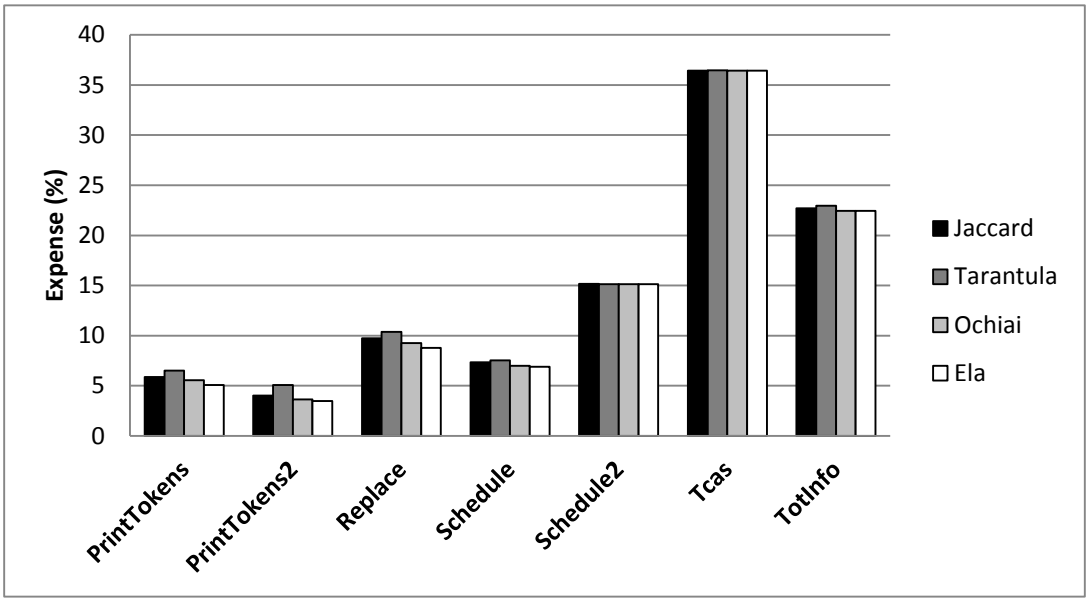

Figure 40 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct Test Suite

Table 33 displays the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct Test Suite. As Table 33 shows, Ela has the lowest expense for the four subject programs and one of the lowest expenses for the three subject programs.

Table 33 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	7.55	8.4	7.19	6.59
PrintTokens2	4.35	5.46	3.94	3.78
Replace	11.33	12.09	10.65	10.02
Schedule	7.65	8.01	7.3	7.22
Schedule2	17.23	17.37	17.09	17.09
Tcas	40.57	40.71	40.57	40.57
TotInfo	24.09	24.67	23.63	23.63

Table 34 shows the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct Test Suite.

Table 34 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	5.86	6.53	5.56	5.08
PrintTokens2	4.02	5.08	3.64	3.48
Replace	9.72	10.38	9.27	8.78
Schedule	7.35	7.52	7	6.91
Schedule2	15.16	15.12	15.12	15.12
Tcas	36.42	36.46	36.42	36.42
TotInfo	22.69	22.97	22.43	22.43

From Table 34, it is clear that Ela technique is superior to three prominent techniques under the specific conditions of our experiments. In the worst case of these experiments, the user still has to inspect on average only 36.42% of the code to find the fault by using Ela technique. In the best case, on average only 3.48% needs to be inspected. Ela technique achieves improvements ranging from 1.25% to 8.7% on average per program over the Ochiai technique which is the second best technique.

Table 34 is that under the specific conditions of our experiments, Ela technique gives a better effectiveness: it always performs at least as good as the other techniques, with an average improvement of 2.81% over the second best technique (Ochiai).

Figure 41 shows LM Improvement over averaged expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct Test Suite.

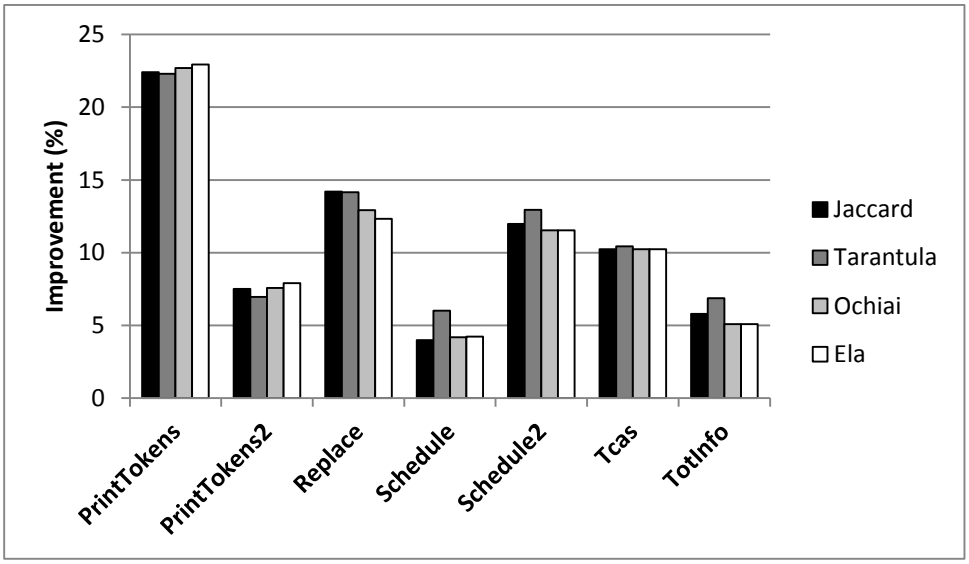

Figure 41 – LM Improvement over Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques for seven programs on Distinct Test Suite

Figure 41 indicates that Ela technique has 10.61% averaged improvement while Jaccard, Tarantula, and Ochiai techniques have 10.87%, 11.38%, and 10.61% averaged improvements respectively on average of all subject programs. There is about 10% averaged improvement of LM strategy for all techniques on Distinct Test Suite.

7.7.2.1. Significance Analysis of Local Maxima with Ela Technique on Distinct Test Suite

We test whether there is a statistically significant improvement of local maxima with Ela or not with Wilcoxon signed rank test at α=0.05 significance level. The null hypothesis H_0 and the alternative hypothesis H_1 are as follows:

H_0: Median Expense Improvement of Local Maxima is equal to zero on Distinct Test Suite.
H_1: Median Expense Improvement of Local Maxima is greater than zero on Distinct Test Suite.

There is a statistically significant improvement of local maxima with Ela (p-value=0.0078) on Distinct Test Suite.

7.7.3. Effective Ranking Strategy on Distinct FminCov Test Suite

In this section, we present the effectiveness of Ela in terms of expense and MRR in the experiments and compare it with those of the three prominent fault localization techniques which are Jaccard, Tarantula, and Ochiai with LM strategy on Distinct FminCov Test Suite.

Figure 42 shows a comparison of the four techniques on the seven subject programs for LM strategy on Distinct FminCov Test Suite. Each version in these seven programs has one fault. We need to examine 4.95% of the source code until we find the faulty statement for the PrintTokens program with all of the four techniques: Ela, Jaccard, Tarantula, and Ochiai.

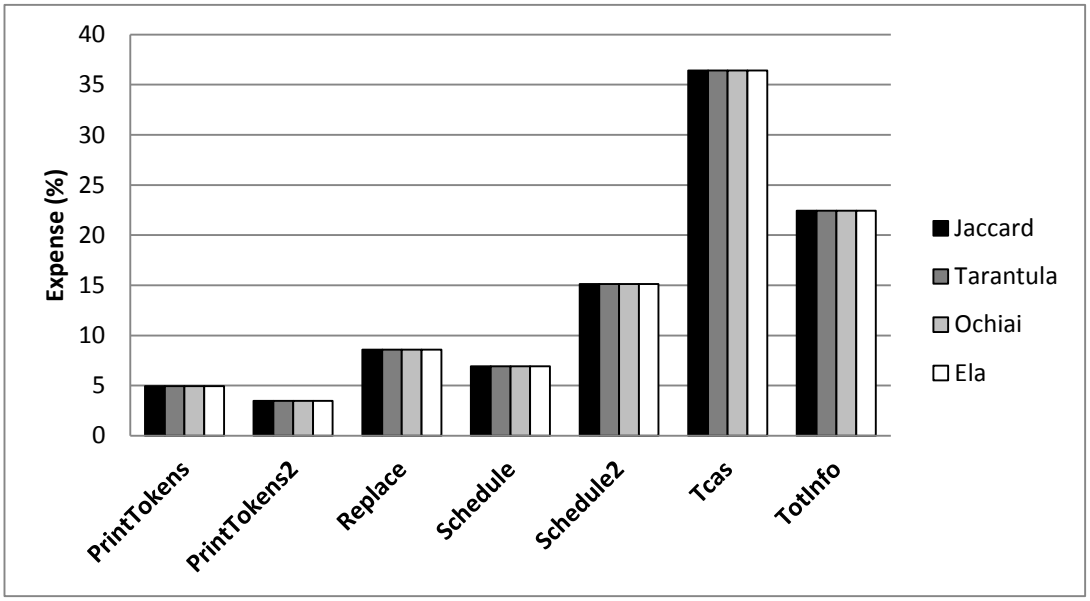

Figure 42 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Test Suite

Table 35 displays the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct FminCov Test Suite. As Table 35 shows, Ela has one of the lowest expenses for each of the programs.

Table 35 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct FminCov Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	6.47	6.47	6.47	6.47
PrintTokens2	3.78	3.78	3.78	3.78
Replace	9.63	9.63	9.63	9.63
Schedule	7.22	7.22	7.22	7.22
Schedule2	17.09	17.09	17.09	17.09
Tcas	40.57	40.57	40.57	40.57
TotInfo	23.61	23.61	23.61	23.61

Table 36 shows the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Test Suite.

Table 36 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	4.95	4.95	4.95	4.95
PrintTokens2	3.48	3.48	3.48	3.48
Replace	8.59	8.59	8.59	8.59
Schedule	6.91	6.91	6.91	6.91
Schedule2	15.12	15.12	15.12	15.12
Tcas	36.42	36.42	36.42	36.42
TotInfo	22.43	22.43	22.43	22.43

From Table 36, it is clear that Ela technique is superior to three prominent techniques under the specific conditions of our experiments. In the worst case of these experiments, the user still has to inspect on average only 36.42% of the code to find the fault by using Ela technique. In the best case, on average only 3.48% needs to be inspected. Ela technique achieves same performance with other three techniques for each of the seven subject programs.

Figure 43 shows LM Improvement over averaged expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct FminCov Test Suite.

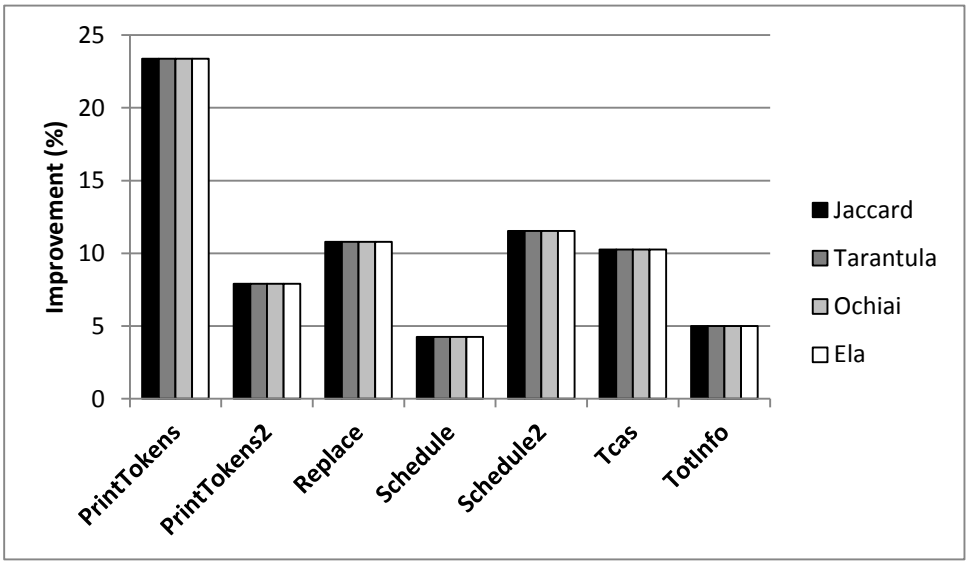

Figure 43 – LM Improvement over Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques for seven programs on Distinct FminCov Test Suite

Figure 43 indicates that Ela and other three techniques have 10.44% averaged improvements on average of all subject programs for Distinct FminCov Test Suite.

7.7.3.1. Significance Analysis of Local Maxima with Ela Technique on Distinct FminCov Test Suite

We test whether there is a statistically significant improvement of local maxima with Ela or not with Wilcoxon signed rank test at α=0.05 significance level. The null hypothesis H_0 and the alternative hypothesis H_1 are as follows:

H_0: Median Expense Improvement of Local Maxima is equal to zero on Distinct FminCov Test Suite.
H_1: Median Expense Improvement of Local Maxima is greater than zero on Distinct FminCov Test Suite.

There is a statistically significant improvement of local maxima with Ela (p-value=0.0078) on Distinct FminCov Test Suite.

7.7.4. Effective Ranking Strategy on Distinct FminCov Cluster Test Suite

In this section, we present the effectiveness of Ela in terms of expense and MRR in the experiments and compare it with those of the three prominent fault localization techniques which are Jaccard, Tarantula, and Ochiai with LM strategy on Distinct FminCov Cluster Test Suite.

Figure 44 shows a comparison of the four techniques on the seven subject programs for LM strategy on Distinct FminCov Cluster Test Suite. Each version in these seven programs has one fault. We need to examine 11% of the source code until we find the faulty statement for the PrintTokens program with all of the four techniques: Ela, Jaccard, Tarantula, and Ochiai.

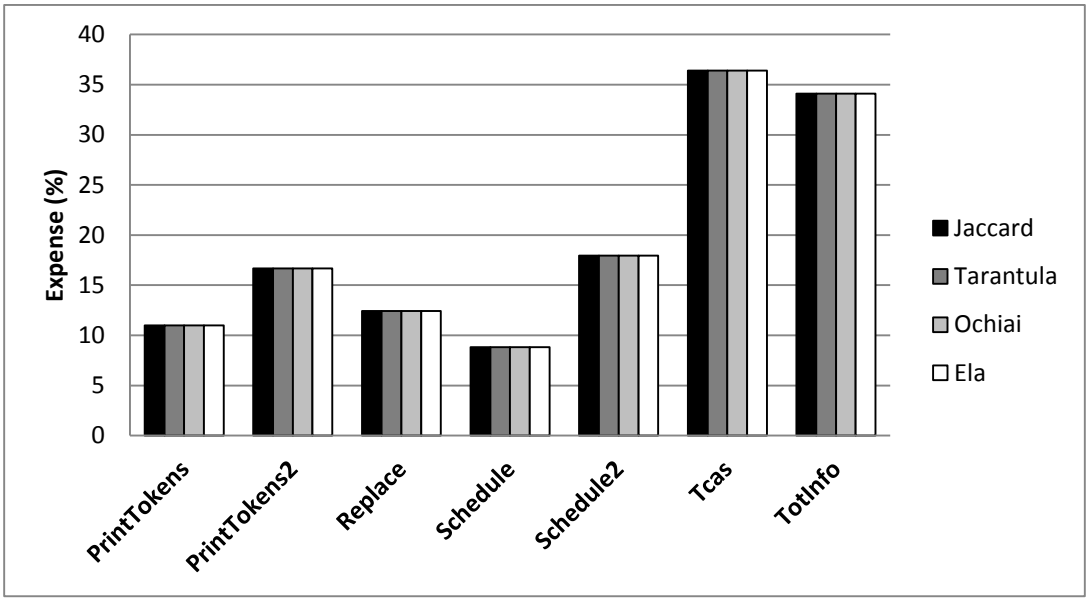

Figure 44 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Cluster Test Suite

Table 37 displays the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct FminCov Cluster Test Suite. As Table 37 shows, Ela has one of the lowest expenses for each of the programs.

Table 37 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct FminCov Cluster Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	13.29	13.29	13.29	13.29
PrintTokens2	17.73	17.73	17.73	17.73
Replace	13.75	13.75	13.75	13.75
Schedule	9.14	9.14	9.14	9.14
Schedule2	19.78	19.78	19.78	19.78
Tcas	40.57	40.57	40.57	40.57
TotInfo	35.48	35.48	35.48	35.48

Table 38 shows the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Cluster Test Suite.

Table 38 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Cluster Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	11	11	11	11
PrintTokens2	16.67	16.67	16.67	16.67
Replace	12.43	12.43	12.43	12.43
Schedule	8.83	8.83	8.83	8.83
Schedule2	17.95	17.95	17.95	17.95
Tcas	36.42	36.42	36.42	36.42
TotInfo	34.1	34.1	34.1	34.1

From Table 38, it is clear that Ela technique is superior to three prominent techniques under the specific conditions of our experiments. In the worst case of these experiments, the user still has to inspect on average only 36.42% of the code to find the fault by using Ela technique. In the best case, on average only 8.83% needs to be inspected. Ela technique achieves same performance with other three techniques for each of the seven subject programs.

Figure 45 shows LM Improvement over averaged expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct FminCov Cluster Test Suite.

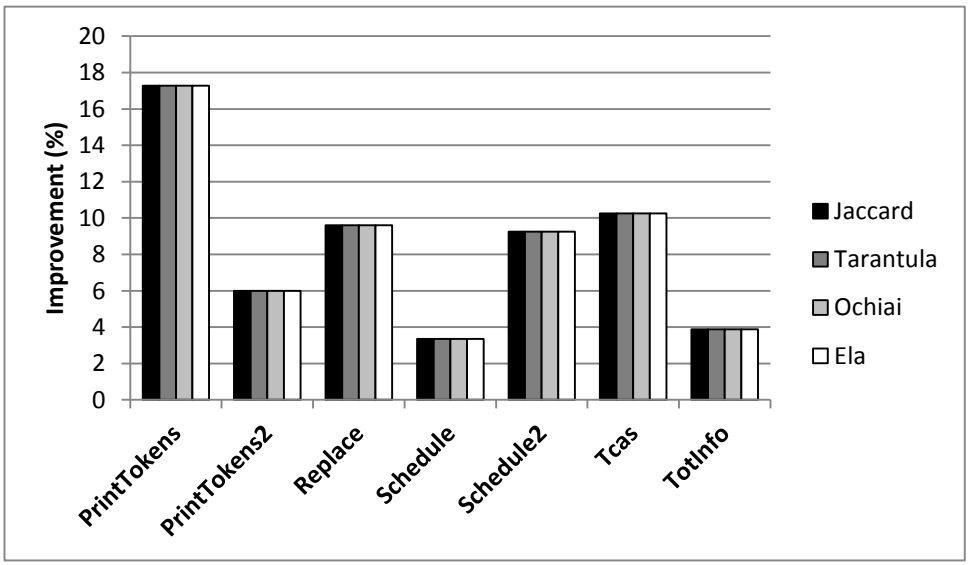

Figure 45 – LM Improvement over Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques for seven programs on Distinct FminCov Cluster Test Suite

Figure 45 indicates that Ela and other three techniques have 8.51% averaged improvements on average of all subject programs for Distinct FminCov Cluster Test Suite.

7.7.4.1. Significance Analysis of Local Maxima with Ela Technique on Distinct FminCov Cluster Test Suite

We test whether there is a statistically significant improvement of local maxima with Ela or not with Wilcoxon signed rank test at α=0.05 significance level. The null hypothesis H_0 and the alternative hypothesis H_1 are as follows:

H_0: Median Expense Improvement of Local Maxima is equal to zero on Distinct FminCov Cluster Test Suite.
H_1: Median Expense Improvement of Local Maxima is greater than zero on Distinct FminCov Cluster Test Suite.

There is a statistically significant improvement of local maxima with Ela (p-value=0.0078) on Distinct FminCov Cluster Test Suite.

7.7.5. Effective Ranking Strategy on Distinct FminCov Classify Test Suite

In this section, we present the effectiveness of Ela in terms of expense and MRR in the experiments and compare it with those of the three prominent fault localization techniques which are Jaccard, Tarantula, and Ochiai with LM strategy on Distinct FminCov Classify Test Suite.

Figure 46 shows a comparison of the four techniques on the seven subject programs for LM strategy on Distinct FminCov Classify Test Suite. Each version in these seven programs has one fault. We need to examine 4.95% of the source code until we find the faulty statement for the PrintTokens program with all of the four techniques: Ela, Jaccard, Tarantula, and Ochiai.

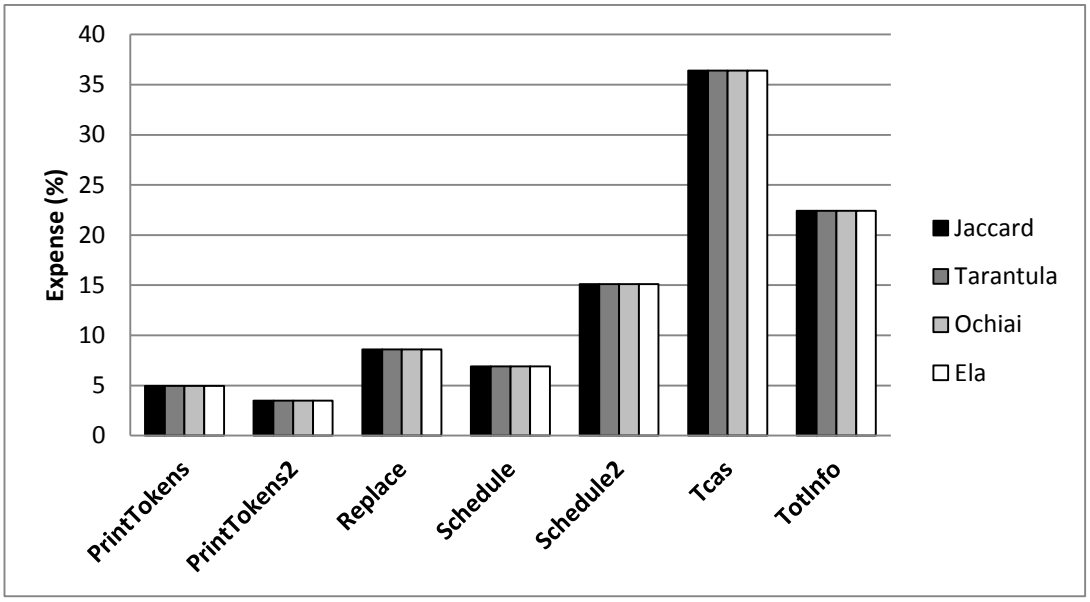

Figure 46 – Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Classify Test Suite

Table 39 displays the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct FminCov Classify Test Suite. As Table 39 shows, Ela has one of the lowest expenses for each of the programs.

Table 39 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct FminCov Classify Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	6.47	6.47	6.47	6.47
PrintTokens2	3.78	3.78	3.78	3.78
Replace	9.63	9.63	9.63	9.63
Schedule	7.22	7.22	7.22	7.22
Schedule2	17.09	17.09	17.09	17.09
Tcas	40.57	40.57	40.57	40.57
TotInfo	23.61	23.61	23.61	23.61

Table 40 shows the averaged expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Classify Test Suite.

Table 40 – Averaged Expenses for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs with LM strategy on Distinct FminCov Classify Test Suite

	Jaccard	Tarantula	Ochiai	Ela
PrintTokens	4.95	4.95	4.95	4.95
PrintTokens2	3.48	3.48	3.48	3.48
Replace	8.59	8.59	8.59	8.59
Schedule	6.91	6.91	6.91	6.91
Schedule2	15.12	15.12	15.12	15.12
Tcas	36.42	36.42	36.42	36.42
TotInfo	22.43	22.43	22.43	22.43

From Table 40, it is clear that Ela technique is superior to three prominent techniques under the specific conditions of our experiments. In the worst case of these experiments, the user still has to inspect on average only 36.42% of the code to find the fault by using Ela technique. In the best case, on average only 3.48% needs to be inspected. Ela technique achieves same performance with other three techniques for each of the seven subject programs.

Figure 47 shows LM Improvement over averaged expense for Jaccard, Tarantula, Ochiai, and Ela techniques on seven programs on Distinct FminCov Classify Test Suite.

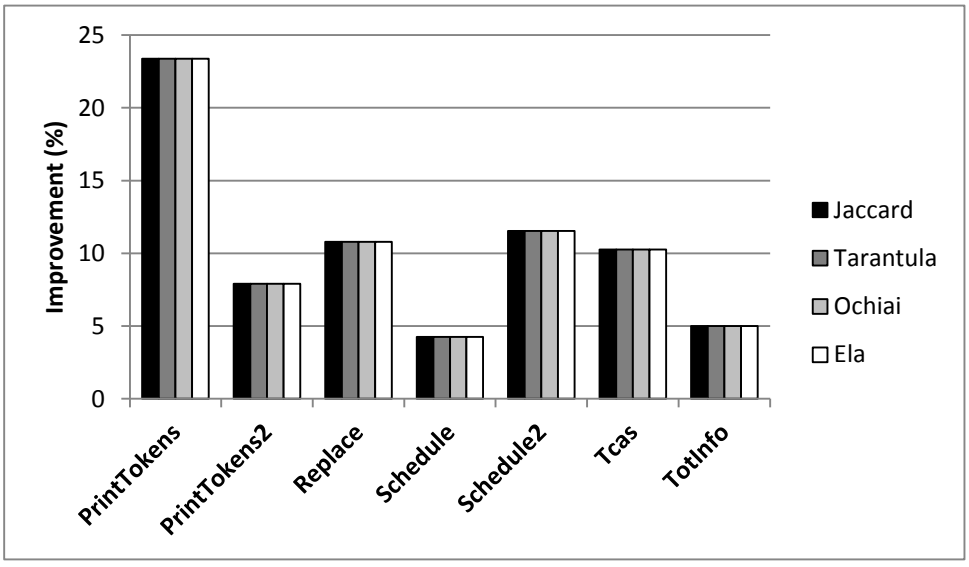

Figure 47 – LM Improvement over Averaged Expense for Jaccard, Tarantula, Ochiai, and Ela techniques for seven programs on Distinct FminCov Classify Test Suite

Figure 47 indicates that Ela and other three techniques have 10.44% averaged improvements on average of all subject programs for Distinct FminCov Classify Test Suite.

7.7.5.1. Significance Analysis of Local Maxima with Ela Technique on Distinct FminCov Classify Test Suite

We test whether there is a statistically significant improvement of local maxima with Ela or not with Wilcoxon signed rank test at α=0.05 significance level. The null hypothesis H_0 and the alternative hypothesis H_1 are as follows:

H_0: Median Expense Improvement of Local Maxima is equal to zero on Distinct FminCov Classify Test Suite.
H_1: Median Expense Improvement of Local Maxima is greater than zero on Distinct FminCov Classify Test Suite.

There is a statistically significant improvement of local maxima with Ela (p-value=0.0078) on Distinct FminCov Classify Test Suite.

7.7.6. Discussion on the Results

The experimental results show that LM strategy achieved about 10% averaged improvements on Redundant, Distinct, Distinct FminCov, and Distinct FminCov Classify test suites for all four techniques while it achieved about 8.5% averaged improvements on Distinct FminCov Cluster test suite for all four techniques. Since fault localization techniques cannot differentiate the faulty statement from the innocent statements due to the limited number of failed test which is not subsumed by any other failed test on Distinct FminCov Cluster test suite, LM strategy achieved relatively smaller averaged improvements in this suite than other four test suites.

Since Ela technique has best or one of the best fault localization effectiveness on five test suites, LM strategy has lowest averaged improvements with Ela technique on these test suites. Moreover, PrintTokens program has the largest averaged block size and Schedule program has the smallest averaged block size. Therefore, LM strategy achieved the highest averaged improvements on PrintTokens program and the lowest averaged improvements on Schedule program.

CHAPTER 8

THREATS TO VALIDITY

In this chapter, we discuss the threats to the validity of our study.

8.1. Internal Validity

Threats to the internal validity can arise from the implementation errors which can affect the results of experiments without the knowledge of researchers and the programs used in the experiments. We used gcov to collect the statement coverage and assume that this tool produces the reliable execution information for the Siemens test suite. This can cause a threat to the internal validity. The feature vectors chosen for clustering and classification contain large number of elements due to the number of statements. This can affect the clustering and the classification processes and could be addressed by using dimensionality reduction techniques. However, we have used these processes only as heuristics.

8.2. External Validity

Threats to the external validity occur when the results of an experiment cannot be generalized to the other situations. The major threat to the external validity is the seven C programs with total of 132 versions in the Siemens suite that we used as subject programs. We have used them to compare our results with related work since it is extensively used in the literature. In addition to the limited number of programs, they are relatively small scale programs and we have used the versions with single fault injected. Therefore, we do not state that our findings generally hold for all the programs from all scales. The reduction strategy is for single fault cases. However, we have discussed that it could be adapted to multi fault case as heuristic which needs to be investigated with further experiments. During the failed test reduction, the test suite of each program contained only one kind of failed test that was subsumed by others. This might not be the case in general. Further empirical evidence with

variety of programs and test suites with different execution profiles are needed to overcome these issues.

8.3. Construct Validity

Threats to the construct validity occur when the metrics used for evaluation do not accurately measure what they intend to measure. The major threat to the construct validity is the metrics used in our evaluations. We used the expense metric as an effectiveness metric to measure and compare the effectiveness of the fault localization techniques. This metric assumes that the programmers inspect the source code in the ranked list of statements, which is generated by the fault localization techniques, and they correctly identify the faulty statements. Although this assumption may not hold for all the cases in software debugging process, this metric is reasonable approximation for effective comparison of fault localization techniques.

8.4. Conclusion Validity

Threats to the conclusion validity occur when a relationship is investigated in the data. Whenever we investigate a relationship, we essentially have either a relationship in our data or not. However, we could lead to an incorrect conclusion in either case. We might conclude that there is a relationship when in fact there is not, or we might infer that there is not a relationship when in fact there is. We assume that there is a relationship between the statement coverage of failed and passed tests and the location of the faulty statement. In addition, we observed this relationship empirically on the seven C program with total of 132 versions in the Siemens suite. The major threat to the conclusion validity is that we can lead to an incorrect conclusion about this relationship in our observations.

CHAPTER 9

CONCLUSIONS

In this dissertation, we presented a fault localization framework that consists of test suite reduction strategies which aim to improve the effectiveness of fault localization, a new fault localization metric, and an effective ranking strategy that improves the ranking of statements.

We propose a new metric for effective statistical fault localization. The intuition is that a faulty statement is more frequently executed by the failed tests and less frequently executed by the passed tests. We also differ from the existing techniques in expressing these frequencies and their combinations. During the comparison, in addition to using the widely accepted expense and code examination effort metrics, we adapt a metric of the information retrieval domain, called mean reciprocal rank (MRR), to assess the overall ranking quality of the SFL techniques. We conducted a number of experiments to measure the effectiveness of Ela and compare it with the three prominent fault localization techniques which are Jaccard, Tarantula, and Ochiai on the Siemens test suite. The proposed technique has a higher ranking than others for 3.4% (4 of 118) of all the versions of the subject programs while it is one of the best performing techniques for 96.6% (114 of 118) of all the versions of the subject programs. Thus, this experimental result answers empirically RQ.1 which inquires if we can define a metric to achieve better fault localization accuracy. The experimental results show the proposed technique outperforms these prominent techniques. As a result, we can conclude that the proposed technique and its different weights for failed and passed tests achieve promising results on the effectiveness of the fault localization techniques in terms of average expense comparing to the three prominent techniques.

We apply equivalent test elimination strategy (Distinct Test Suite Strategy) to achieve more accurate fault localization. If two tests have executed the set of same statements, then we

consider them as equivalent tests with respect to their statement coverage. We assume that there are several bad tests in the test suite: bad passed tests and bad fail tests. A bad passed test is defined as “the test that passes even if it executes the faulty statement”. A bad fail test is defined as “the test that executes too many innocent statements”. We empirically show that increasing the tests that are equivalent to a bad test case affects the result of the fault localization. Since it is not known that whether a test is good test or not, the existence of the equivalent tests can cause a bias in the ranking of statements according to their suspiciousness values during the process of statistical fault localization. Therefore, we select the safe side and eliminate the equivalent tests from both passed and failed tests in the experiments. We conducted a number of experiments to measure the effectiveness of Distinct Test Suite Strategy for Jaccard, Tarantula, Ochiai and Ela on the Siemens test suite. This strategy achieves on average 99.5% test size reduction. As a result, we can conclude that the proposed test suite reduction strategy significantly reduce the test suite; therefore, it will reduce the effort significantly too. This experimental result empirically answers RQ.2 which asks the effect of equivalent tests in the fault localization effectiveness. Moreover, the proposed technique has a higher ranking than others for 26.3% (31 of 118) of all the versions of the subject programs while it is one of the best performing techniques for 73.7% (87 of 118) of all the versions of the subject programs with this Distinct Test Suite Strategy. Thus, this experimental result answers empirically RQ.1 which inquires if we can define a metric to achieve better fault localization accuracy.

We propose a new test suite reduction strategy to reduce the effort for the fault localization by reducing the test suite size. Different from the literature, we propose to eliminate the failed tests that may mislead the SFL to assign higher suspiciousness values to innocent program elements (FminCov Test Suite Strategy). We empirically show the effect of this strategy on three popular SFL (Tarantula, Jaccard, and Ochiai) by using the Siemens suite. We conducted a number of experiments to measure the effectiveness of FminCov Test Suite Strategy for Jaccard, Tarantula, Ochiai and Ela on the Siemens test suite. This strategy achieves on average 10.1% test size reduction. As a result, we can conclude that the proposed test suite reduction strategy considerably reduce the test suite; therefore, it will reduce the effort considerably too. Hence, this experimental result answers empirically RQ.3.1 which asks what type of tests affect test suite quality.

To address RQ.3.2, we investigate two kinds of reductions of the passed tests in combination with the FminCov Test Suite Strategy. RQ.3.2 asks if clustering and/or classification of failed and passed tests can result in a subset of test suite that increase the fault localization

effectiveness. In the first passed test case elimination strategy, we aim to eliminate the passed tests that are close to the eliminated failed tests, similar to Masri and Assi (2014). The intuition is to remove the test that have executed the faulty statement but still pass (Classification Strategy). We classify the passed tests into the eliminated failed tests and the remaining failed tests by using KNN classification algorithm and then select the class that contains the remaining failed tests and remove other class. We conducted a number of experiments to measure the effectiveness of Classification strategy for Jaccard, Tarantula, Ochiai and Ela on the Siemens test suite. This strategy achieves on average 30.3% test size reduction. As a result, we can conclude that the proposed test suite reduction strategy greatly reduce the test suite; therefore, it will reduce the effort greatly too. Hence, this experimental result answers empirically RQ.3.2. In the second passed test case elimination strategy, similar to Dandan et al. (2013), we aim to eliminate the passed tests that are not very close to the remaining failed tests (Clustering Strategy). The intuition is that of delta debugging (Zeller, 1999) which states that a passing run closest to a failing run contains the most information. We cluster the remaining failed tests and all the passed tests into subsets by using a hierarchical clustering algorithm, specifically Agglomerative clustering, and then select the subset that contains the remaining failed tests. This strategy achieves on average 34.1% test size reduction. As a result, we can conclude that the proposed test suite reduction strategy greatly reduce the test suite; therefore, it will reduce the effort greatly too. Hence, this experimental result answers empirically RQ.3.2.

In our experiments, we examine all three test suite reduction strategies and shows that all the reduction strategies result in significant reductions in the size of tests. Among all three, the failed test elimination strategy results in the best improvement but the elimination of the passed tests similar to the eliminated failed tests (Classification Strategy) results in quite comparable results to failed test elimination strategy. Removing the passed tests similar to the remaining failed tests results in high reductions in the size of test suite (up to 81%). We conducted a number of experiments to measure the effectiveness of Clustering strategy for Jaccard, Tarantula, Ochiai and Ela on the Siemens test suite. Best of these test suite reduction strategies achieves an improvement up to 1.7% in Jaccard, up to 2.46% in Tarantula, up to 1.01% in Ochiai, and up to 0.38% in Ela in terms of average expense.

We propose a new effective ranking strategy for improving the ranking of statements (Local Maxima Strategy) in order to improve the effectiveness and therefore decrease the effort for the fault localization. Instead of serving all the statements with their suspiciousness ranks to the software developers, we aim to serve only the statements which are the local maximum

in its 1-nearest neighborhood. This strategy decreases the number of statements that software developers must inspect to locate the fault. It assumes that the innocent statements near to the faulty statement are likely to be assigned with high suspiciousness values and should be eliminated from the list of suspicious statements. We conducted a number of experiments to measure the effectiveness of Local Maxima strategy for Jaccard, Tarantula, Ochiai and Ela on the Siemens test suite. This strategy achieves an improvement 10.54% in Jaccard, 10.47% in Tarantula, 10.74% in Ochiai, and 10.88% in Ela in terms of average expense. As a result, we can conclude that the proposed technique achieve considerable improvement over the effectiveness of the fault localization techniques in terms of average expense. This experimental result answers empirically RQ.4 which asks if we can implement a post processing technique to improve the fault localization accuracy based on suspiciousness values of statements.

CHAPTER 10

FUTURE WORK

This chapter discusses the ideas on extending our work and on future research directions to be investigated.

10.1. Application of Ela to Different Test Suites from Different Scales

Siemens test suite used in our study has a limited number of computer programs. We plan to evaluate the robustness of Ela technique on different kinds of computer programs. The programs written in sequential and procedural programming languages such as C language are chosen as the subject programs from Siemens test suite. We will perform experiments on the computer programs written in object oriented programming languages such as Java and C# languages. In addition, the subject programs are relatively small scale programs. Therefore, we plan to perform experiments on the large scale computer programs which contain real life software faults.

In addition, we can measure the effectiveness of Ela on different kinds of programs by classifying the versions of different programs into different categories according to the nature of their fault types. We plan to investigate the relation of fault types and fault localization power of Ela technique by performing several experiments on different kinds of programs.

10.2. Application of Ela with Different Coverage Entities

This dissertation has proposed a new fault localization metric applied on the statement coverage of tests. This technique can be applied to a number of different coverage entities such as classes, methods, blocks, branches, predicates etc. Future researchers may evaluate the effectiveness of Ela technique with other coverage entities. The type of coverage entity may affect the effectiveness of Ela according to the types of faults used. Future researchers

may explore the correlation between the fault types and the coverage entities to find best coverage entity.

10.3. Investigating Optimal Percentage of Failed Tests in Test Suites

The percentage of the failed tests in the Siemens test suite is relatively small. We can achieve higher percentage of the failed tests in other test suites and investigate the optimal percentage of the failed tests in the test suites. Therefore, we will carry out experiments to investigate the optimal percentage of the failed tests in other test suites. Moreover, test case generation techniques in the literature (Rayadurgam & Heimdahl, 2001; Artho et al., 2003; Papadakis & Malevris, 2010) can be applied to create additional failed tests and increase their percentage in the test suites.

10.4. Application of Ela to Multiple Faults

We have used the versions of the subject programs with single fault in our study. Although we have stated that Ela technique can be applied to the multiple faults, we want to see more experimental evidences to support our conclusion. Therefore, we will conduct new experiments on the computer programs with multiple faults and evaluate the effectiveness of Ela technique.

10.5. Designing User Studies on Focus Groups

An empirical justification of the evaluation metrics used in our study indicates a confidence that Ela technique provides an evidence of good effectiveness. The intuition behind the hypothetical idea of a perfect debugging, which is the breadth first search with terminating when a faulty line is encountered, seems reasonable. However, it is highly desirable to show the direct correlation between the effectiveness of Ela technique under the evaluation metrics and under the actual user debugging experience. Therefore, we plan to design several user studies on focus group of software developers to evaluate effectiveness of Ela technique and compare with the prominent techniques in the literature on the subject programs.

REFERENCES

[1] Abreu, R., Zoeteweij, P., & Van Gemund, A. J. (2006, December). An evaluation of similarity coefficients for software fault localization. In *Dependable Computing, 2006. PRDC'06. 12th Pacific Rim International Symposium on* (pp. 39-46). IEEE.

[2] Abreu, R., Zoeteweij, P., & Van Gemund, A. J. (2007, September). On the accuracy of spectrum-based fault localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007. TAICPART-MUTATION 2007* (pp. 89-98). IEEE.

[3] Agarwal, P., & Agrawal, A. P. (2014). Fault-localization techniques for software systems: a literature review. *ACM SIGSOFT Software Engineering Notes*, *39*(5), 1-8.

[4] Aggarwal, C. C., & Reddy, C. K. (Eds.). (2013). An Introduction to Cluster Analysis. *Data clustering: algorithms and applications* (pp. 5-7). CRC Press.

[5] Agrawal, H. (1991). *Towards automatic debugging of computer programs* (Doctoral dissertation, Purdue University).

[6] Agrawal, H., De Millo, R. A., & Spafford, E. H. (1991). An execution-backtracking approach to debugging. *Software, IEEE*, *8*(3), 21-26.

[7] Agrawal, H., Horgan, J., London, S., & Wong, W. (1995). Fault localization using execution slices and dataflow tests. *Proceedings of IEEE Software Reliability Engineering*, 143-151.

[8] Alipour, M. A. (2012). *Automated fault localization techniques: a survey*. Technical report, Oregon State University.

[9] Artho, C., Drusinksy, D., Goldberg, A., Havelund, K., Lowry, M., Pasareanu, C., & Visser, W. (2003, January). Experiments with test case generation and runtime analysis. In *Abstract State Machines 2003* (pp. 87-108). Springer Berlin Heidelberg.

[10] Artzi, S., Dolby, J., Tip, F., & Pistoia, M. (2012). Fault localization for dynamic web applications. *Software Engineering, IEEE Transactions on*, *38*(2), 314-335.

[11] Augusteijn, L. (2002). Front: a front-end generator for Lex, Yacc and C, *Release 1.0, Retrieved from: http://www.extra.research.philips.com/ist, Philips Research Laboratories, Eindhoven, Netherlands*.

[12] Baah, G. K., Podgurski, A., & Harrold, M. J. (2011, September). Mitigating the confounding effects of program dependences for effective fault localization. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering* (pp. 146-156). ACM.

[13] Balzer, R. M. (1969, May). EXDAMS: extendable debugging and monitoring system. In *Proceedings of the May 14-16, 1969, spring joint computer conference* (pp. 567-580). ACM.

[14] Baudry, B., Fleurey, F., & Le Traon, Y. (2006, May). Improving test suites for efficient fault localization. In *Proceedings of the 28th international conference on Software engineering* (pp. 82-91). ACM.

[15] Brase, C. H., & Brase, C. P. (2011). *Understandable statistics: concepts and methods*. Cengage Learning.

[16] Chan, W. K., & Cai, Y. (2013). In quest of the science in statistical fault localization. *Software: Practice and Experience*, *43*(8), 971-987.

[17] Cleve, H., & Zeller, A. (2005, May). Locating causes of program failures. In *Proceedings of the 27th international conference on Software engineering* (pp. 342-351). ACM.

[18] Dallmeier, V., Lindig, C., & Zeller, A. (2005, September). Lightweight bug localization with AMPLE. In *Proceedings of the sixth international symposium on Automated analysis-driven debugging* (pp. 99-104). ACM.

[19] Dandan, G., Tiantian, W., Xiaohong, S., & Peijun, M. (2013). A test-suite reduction approach to improving fault-localization effectiveness. *Computer Languages, Systems & Structures*, *39*(3), 95-108.

[20] Dickinson, W., Leon, D., & Podgurski, A. (2001a, July). Finding failures by cluster analysis of execution profiles. In *Proceedings of the 23rd international conference on Software engineering* (pp. 339-348). IEEE Computer Society.

[21] Dickinson, W., Leon, D., & Podgurski, A. (2001b, September). Pursuing failure: the distribution of program failures in a profile space. In *ACM SIGSOFT Software Engineering Notes* (Vol. 26, No. 5, pp. 246-255). ACM.

[22] DiGiuseppe, N., & Jones, J. A. (2011, July). On the influence of multiple faults on coverage-based fault localization. In *Proceedings of the 2011 international symposium on software testing and analysis* (pp. 210-220). ACM.

[23] Farjo, J., & Masri, W. (2014, March). Weighted Execution Profiles for Software Testing. In *Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on* (pp. 298-301). IEEE.

[24] Farris, J. S. (1969). On the cophenetic correlation coefficient. *Systematic Biology*, *18*(3), 279-285.

[25] Hao, D., Zhang, L., Zhong, H., Mei, H., & Sun, J. (2005a, September). Eliminating harmful redundancy for testing-based fault localization using test suite reduction: An experimental study. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on* (pp. 683-686). IEEE.

[26] Hao, D., Pan, Y., Zhang, L., Zhao, W., Mei, H., & Sun, J. (2005b, November). A similarity-aware approach to testing based fault localization. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering* (pp. 291-294). ACM.

[27] Hao, D., Xie, T., Zhang, L., Wang, X., Sun, J., & Mei, H. (2010). Test input reduction for result inspection to facilitate fault localization. *Automated software engineering*, *17*(1), 5-31.

[28] Harrold, M. J., Gupta, R., & Soffa, M. L. (1993). A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *2*(3), 270-285.

[29] Hutchins, M., Foster, H., Goradia, T., & Ostrand, T. (1994, May). Experiments of the effectiveness of dataflow-and controlflow-based test adequacy criteria. In *Proceedings of the 16th international conference on Software engineering* (pp. 191-200). IEEE Computer Society Press.

[30] Javac. (2014). *Retrieved from: http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javac.html*

[31] Jeffrey, D., Gupta, N., & Gupta, R. (2008, July). Fault localization using value replacement. In *Proceedings of the 2008 international symposium on Software testing and analysis* (pp. 167-178). ACM.

[32] Jigsaw. (2014). *Retrieved from: http://www.w3.org/Jigsaw/*

[33] Jikes. (2014). *Retrieved from: http://jikes.sourceforge.net/*

[34] Jones, J. A., Harrold, M. J., & Stasko, J. (2002, May). Visualization of test information to assist fault localization. In *Proceedings of the 24th international conference on Software engineering* (pp. 467-477). ACM.

[35] Jones, J. A., & Harrold, M. J. (2005, November). Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering* (pp. 273-282). ACM.

[36] Jones, J. A., Bowring, J. F., & Harrold, M. J. (2007, July). Debugging in parallel. In *Proceedings of the 2007 international symposium on Software testing and analysis* (pp. 16-26). ACM.

[37] Jones, J. A. (2008). *Semi-Automatic Fault Localization* (Doctoral dissertation, Georgia Institute of Technology).

[38] JTidy. (2014). *Retrieved from: http://jtidy.sourceforge.net/*

[39] Ju, X., Jiang, S., Chen, X., Wang, X., Zhang, Y., & Cao, H. (2014). HSFal: Effective fault localization using hybrid spectrum of full slices and execution slices. *Journal of Systems and Software*, *90*, 3-17.

[40] Kim, J., & Lee, E. (2014, February). Empirical evaluation of existing algorithms of spectrum based fault localization. In *Information Networking (ICOIN), 2014 International Conference on* (pp. 346-351). IEEE.

[41] Lange, M. M. (2014). *A Comparison of Estimators for Respondent-Driven Sampling* (Master's thesis, University of California).

[42] Liblit, B., Aiken, A., Zheng, A. X., & Jordan, M. I. (2003). Bug isolation via remote program sampling. *ACM SIGPLAN Notices*, *38*(5), 141-154.

[43] Liblit, B., Naik, M., Zheng, A. X., Aiken, A., & Jordan, M. I. (2005, June). Scalable statistical bug isolation. In *ACM SIGPLAN Notices* (Vol. 40, No. 6, pp. 15-26). ACM.

[44] Liu, C., Yan, X., Fei, L., Han, J., & Midkiff, S. P. (2005). SOBER: statistical model-based bug localization. *ACM SIGSOFT Software Engineering Notes*, *30*(5), 286-295.

[45] Liu, C., Fei, L., Yan, X., Han, J., & Midkiff, S. P. (2006). Statistical debugging: A hypothesis testing-based approach. *Software Engineering, IEEE Transactions on*, *32*(10), 831-848.

[46] Liu, C., & Han, J. (2006, November). Failure proximity: a fault localization-based approach. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering* (pp. 46-56). ACM.

[47] Maheswari, G., & Venkatesakumar, V. (2013, May). Fault Localization for Dynamic Web Application. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, 2(3), 71-77.

[48] Mao, X., Lei, Y., Dai, Z., Qi, Y., & Wang, C. (2014). Slice-based statistical fault localization. *Journal of Systems and Software*, *89*, 51-62.

[49] Masri, W., & Assi, R. A. (2014). Prevalence of coincidental correctness and mitigation of its impact on fault localization. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *23*(1), 8.

[50] Naish, L., Lee, H. J., & Ramamohanarao, K. (2011). A model for spectra-based software diagnosis. *ACM Transactions on software engineering and methodology (TOSEM)*, *20*(3), 11.

[51] Newman, M. (2002). Software errors cost US economy 59.5 billion annually, NIST assesses technical needs of industry to improve software-testing. *Press Release, http://www.nist.gov/public_affairs/releases*, (02), 10.

[52] Nienhuys, H. W., & Nieuwenhuizen, J. (2003, May). LilyPond, a system for automated music engraving. In *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)* (pp. 167-172).

[53] Ochiai, A. (1957). Zoogeographic studies on the soleoid fishes found in Japan and its neighbouring regions. *Bull. Jpn. Soc. Sci. Fish*, *22*(9), 526-530.

[54] Papadakis, M., & Malevris, N. (2010, November). Automatic mutation test case generation via dynamic symbolic execution. In *Software reliability engineering (ISSRE), 2010 IEEE 21st international symposium on* (pp. 121-130). IEEE.

[55] Parsa, S., Vahidi-Asl, M., Arabi, S., & Minaei-Bidgoli, B. (2009). Software fault localization using elastic net: A new statistical approach. In *Advances in Software Engineering* (pp. 127-134). Springer Berlin Heidelberg.

[56] Podgurski, A., Masri, W., McCleese, Y., Wolff, F. G., & Yang, C. (1999). Estimation of software reliability by stratified sampling. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *8*(3), 263-283.

[57] Podgurski, A., Leon, D., Francis, P., Masri, W., Minch, M., Sun, J., & Wang, B. (2003, May). Automated support for classifying software failure reports. In *Software Engineering, 2003. Proceedings. 25th International Conference on* (pp. 465-475). IEEE.

[58] Qi, Y., Mao, X., Lei, Y., & Wang, C. (2013, July). Using automated program repair for evaluating the effectiveness of fault localization techniques. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis* (pp. 191-201). ACM.

[59] Rayadurgam, S., & Heimdahl, M. P. (2001). Coverage based test-case generation using model checkers. In *Engineering of Computer Based Systems, 2001. ECBS 2001. Proceedings. Eighth Annual IEEE International Conference and Workshop on the* (pp. 83-91). IEEE.

[60] Renieres, M., & Reiss, S. P. (2003, October). Fault localization with nearest neighbor queries. In *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on* (pp. 30-39). IEEE.

[61] Ridgeway, M. (2004). Using code coverage tools in the Linux kernel.

[62] Rothermel, G., Harrold, M. J., Ostrin, J., & Hong, C. (1998, November). An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *Software Maintenance, 1998. Proceedings. International Conference on* (pp. 34-43). IEEE.

[63] Sahoo, S. K., Criswell, J., Geigle, C., & Adve, V. (2013). Using likely invariants for automated software fault localization. *ACM SIGARCH Computer Architecture News*, *41*(1), 139-152.

[64] Shapiro, E. Y. (1983). *Algorithmic Program Debugging.* Cambridge, MA, USA: MIT Press.

[65] Shin, W. H., Muthaiyah, S., & Raman, M. (2012, December). Green Evaluation Metrics and Software Tool for Data Center. *Advances in Environment, Computational Chemistry and Bioscience*, (pp. 25-30).

[66] Software-artifact infrastructure repository (SIR). (2014).

Retrieved from: http://sir.unl.edu/php/previewfiles.php.

[67] Srivastav, M., Singh, Y., & Chauhan, D. S. (2010). Faulty Slice Distribution using Complexity Estimation for Debugging in Parallel. *International Journal of Computer Applications (0975–8887), 7*(1).

[68] Stallman, R. M. (2009). *Using the GNU Compiler Collection: A GNU Manual for GCC Version 4.3. 3*. CreateSpace.

[69] Su, X. H., Gong, D. D., Wang, T. T., & Ma, P. J. (2014, July). A Survey of Automated Software Fault Localization Approach. In *Applied Mechanics and Materials* (Vol. 556, pp. 6102-6105).

[70] Tan, P. N., Kumar, V., & Srivastava, J. (2002, July). Selecting the right interestingness measure for association patterns. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 32-41). ACM.

[71] Tomcat. (2014). *Retrieved from:* http://tomcat.apache.org/

[72] Vallée-Rai, R., Co, P., Gagnon, E., Hendren, L., Lam, P., & Sundaresan, V. (1999, November). Soot-a Java bytecode optimization framework. In *Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research* (p. 13). IBM Press.

[73] Vidács, L., Beszédes, Á., Tengeri, D., Siket, I., & Gyimóthy, T. (2014, February). Test suite reduction for fault detection and localization: A combined approach. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on* (pp. 204-213). IEEE.

[74] Voorhees, E. M. (1999, November). The TREC-8 Question Answering Track Report. In *TREC* (Vol. 99, pp. 77-82).

[75] Wong, W. E., & Debroy, V. (2009). A survey of software fault localization. *Department of Computer Science, University of Texas at Dallas, Tech. Rep. UTDCS-45-09*.

[76] Wong, W. E., Debroy, V., & Xu, D. (2012). Towards better fault localization: A crosstab-based statistical approach. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 42*(3), 378-396.

[77] Xie, X., Wong, W. E., Chen, T. Y., & Xu, B. (2010, February). Spectrum-based fault localization without test oracles. In *proceedings of the 11th International Conference on Quality Software (QSIC'11)* (pp. 1-10).

[78] χSuds. (1998). Software Understanding System User's Manual, *Release 1.2, Retrieved from:* https://www.cs.purdue.edu/homes/apm/foundationsBook/Labs/coverage/xsuds.pdf.

[79] Xu, J., Chan, W. K., Zhang, Z., Tse, T. H., & Li, S. (2011, July). A dynamic fault localization technique with noise reduction for java programs. In *Quality Software (QSIC), 2011 11th International Conference on* (pp. 11-20). IEEE.

[80] Xu, J., Zhang, Z., Chan, W. K., Tse, T. H., & Li, S. (2013). A general noise-reduction framework for fault localization of Java programs. *Information and Software Technology*, *55*(5), 880-896.

[81] Yoo, S., Harman, M., & Clark, D. (2013). Fault localization prioritization: Comparing information-theoretic and coverage-based approaches. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *22*(3), 19.

[82] Yu, Y., Jones, J. A., & Harrold, M. J. (2008, May). An empirical study of the effects of test-suite reduction on fault localization. In *Proceedings of the 30th international conference on Software engineering* (pp. 201-210). ACM.

[83] Zeller, A. (1999, January). Yesterday, my program worked. Today, it does not. Why?. In *Software Engineering—ESEC/FSE'99* (pp. 253-267). Springer Berlin Heidelberg.

[84] Zhang, Z., Chan, W. K., Tse, T. H., Jiang, B., & Wang, X. (2009, August). Capturing propagation of infected program states. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering* (pp. 43-52). ACM.

[85] Zhang, Z., Chan, W. K., Tse, T. H., Yu, Y. T., & Hu, P. (2011). Non-parametric statistical fault localization. *Journal of Systems and Software*, *84*(6), 885-905.

A. WILCOXON SIGNED RANK TESTS

Table 41 – Wilcoxon signed rank test for the significance test of Ela and Jaccard on seven programs with Redundant Test Suite

Programs	Ela & Jaccard Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	0.125	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	0.016	Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	$6.1x10^{-5}$	Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	0.125	Fail to Reject H_0

Table 42 – Wilcoxon signed rank test for the significance test of Ela and Tarantula on seven programs with Redundant Test Suite

Programs	Ela & Tarantula Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	0.125	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	0.008	Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	$6.1x10^{-5}$	Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	0.125	Fail to Reject H_0

Table 43 – Wilcoxon signed rank test for the significance test of Ela and Ochiai on seven programs with Redundant Test Suite

Programs	Ela & Ochiai Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	0.125	Fail to Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	0.5	Fail to Reject H_0

Table 44 – Wilcoxon signed rank test for the significance test of Ela and Jaccard on seven programs with Distinct Test Suite

Programs	Ela & Jaccard Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	0.125	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	0.031	Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	$7.5x10^{-9}$	Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	0.063	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	0.125	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	$9.8x10^{-4}$	Reject H_0

Table 45 – Wilcoxon signed rank test for the significance test of Ela and Tarantula on seven programs with Distinct Test Suite

Programs	Ela & Tarantula Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	0.031	Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	$2.0x10^{-3}$	Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	$7.5x10^{-9}$	Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	0.063	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	0.125	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	0.016	Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	$2.4x10^{-4}$	Reject H_0

Table 46 – Wilcoxon signed rank test for the significance test of Ela and Ochiai on seven programs with Distinct Test Suite

Programs	Ela & Ochiai Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	0.25	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	0.5	Fail to Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	$7.5x10^{-9}$	Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	0.5	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0

Table 47 – Wilcoxon signed rank test for the significance test of Ela and Jaccard on seven programs with Distinct FminCov Test Suite

Programs	Ela & Jaccard Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0

Table 48 – Wilcoxon signed rank test for the significance test of Ela and Tarantula on seven programs with Distinct FminCov Test Suite

Programs	Ela & Tarantula Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0

Table 49 – Wilcoxon signed rank test for the significance test of Ela and Ochiai on seven programs with Distinct FminCov Test Suite

Programs	Ela & Ochiai Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0

Table 50 – Wilcoxon signed rank test for the significance test of Ela and Jaccard on seven programs with Distinct FminCov Cluster Test Suite

Programs	Ela & Jaccard Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0

Table 51 – Wilcoxon signed rank test for the significance test of Ela and Tarantula on seven programs with Distinct FminCov Cluster Test Suite

Programs	Ela & Tarantula Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0

Table 52 – Wilcoxon signed rank test for the significance test of Ela and Ochiai on seven programs with Distinct FminCov Cluster Test Suite

Programs	Ela & Ochiai Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0

Table 53 – Wilcoxon signed rank test for the significance test of Ela and Jaccard on seven programs with Distinct FminCov Classify Test Suite

Programs	Ela & Jaccard Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Jaccard, Expense) H_1:Med(Ela, Expense) < Med(Jaccard, Expense)	0.05	1	Fail to Reject H_0

Table 54 – Wilcoxon signed rank test for the significance test of Ela and Tarantula on seven programs with Distinct FminCov Classify Test Suite

Programs	Ela & Tarantula Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Tarantula, Expense) H_1:Med(Ela, Expense) < Med(Tarantula, Expense)	0.05	1	Fail to Reject H_0

Table 55 – Wilcoxon signed rank test for the significance test of Ela and Ochiai on seven programs with Distinct FminCov Classify Test Suite

Programs	Ela & Ochiai Comparison	Alpha	P-value	Result
PrintTokens	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
PrintTokens2	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Replace	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Schedule	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Schedule2	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
Tcas	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0
TotInfo	H_0:Med(Ela, Expense)=Med(Ochiai, Expense) H_1:Med(Ela, Expense) < Med(Ochiai, Expense)	0.05	1	Fail to Reject H_0

B. FAULT TYPES AND THEIR DIFFICUILTIES

Table 56 – Fault types and their difficulties in seven programs of Siemens test suite

Fault Type	Difficuilty
Add [if, else-if, else] condition	Low
Change [if, else-if, else] condition	Low
Comment or Delete [if, else-if, else] condition	Low
Add [if, else-if, else] block	Low
Change [if, else-if, else] block	Low
Comment or Delete [if, else-if, else] block	Low
Add case condition	Low
Change case condition	Low
Comment or Delete case condition	Low
Add case block	Low
Change case block	Low
Comment or Delete case block	Low
Add [for, while] condition	Low
Change [for, while] condition	Low
Comment or Delete [for, while] condition	Low
Add [for, while] block	Low
Change [for, while] block	Low
Comment or Delete [for, while] block	Low
Add return value	Medium
Change return value	Medium
Comment or Delete return value	Medium
Add method call	High
Change method call	High
Comment or Delete method call	High
Add method body	High
Change method body	High
Comment or Delete method body	High
Add variable assignment	High
Change variable assignment	High
Comment or Delete variable assignment	High
Add variable initialization	High
Change variable initialization	High
Comment or Delete variable initialization	High
Add lines of code	High
Change lines of code	High

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name	: Bayraktar, Özkan
Nationality	: Turkish (TC)
Date and Place of Birth	: 30 September 1979, Ankara
Marital Status	: Married
Phone	: +90 532 4036135
Email	: ozkanbayraktar@gmail.com

EDUCATION

Degree	Institution	Graduation Year
PhD.	Metu – Information Systems	2007 – present
Ms.	Metu – Information Systems	2004 – 2007
Bs.	Metu – Statistics	1998 – 2003
High School	Ankara Anıttepe High School	1994 – 1997

WORK EXPERIENCE

Year	Place	Enrollment
2004 – 2012	METU – Informatics Institute / TURKEY	Research Assistant
2012 – 2014	GCA L.C. / TURKEY	Senior Software Specialist
2014 – present	Biznet I.C. / TURKEY	Senior Software Specialist

FOREIGN LANGUAGES

Advanced English