AN IMPROVED HIERARCHICAL CLUSTERING COMBINATION APPROACH FOR SOFTWARE MODULARIZATION

RASHID NASEEM

UNIVERSITI TUN HUSSEIN ONN MALAYSIA
AN IMPROVED HIERARCHICAL CLUSTERING COMBINATION APPROACH
FOR SOFTWARE MODULARIZATION

RASHID NASEEM

A thesis submitted in
fulfillment of the requirement for the award of the
Doctor of Philosophy in Information Technology

Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

JANUARY 2017
To my family.
ACKNOWLEDGEMENT

First, I want to thank ALLAH Almighty for giving me the strength and courage to accomplish my goal. HE has been the biggest source of strength for me.

I would like to express my deepest gratitude to my supervisor Prof. Dr. Mustafa Mat Deris. It has been an honor to be his PhD student. I am grateful for all his contributions of time, ideas, and knowledge to make my research experience productive and exciting.

I benefited from the expertise and support of Dr. Onaiza Mabool of Quaid-I-Azam University, Pakistan, in all aspects of my research work. Her guidance and instructions were particularly valuable during preparing research papers and the thesis.

I would like to thank University Tun Hussein Onn Malaysia (UTHM) for supporting this research under Graduate Researcher Incentive Grant (GIPS) Vote No. U063.

I am grateful to Siraj Muhammad and Abdul Qadoos Abbasi who shared my interests and helped me a lot to clarify my views through conversations and implementation concerning my research and provide me the test systems for my research.

I would like to show my appreciation and pleasure to all my friends from Nigeria, Somalia, Iraq, Yemen, Malaysia, and Pakistan, especially the friends of Parit Raja (our house on rent in Batu Pahat, Malaysia). The loving and caring friends of Parit Raja: Zinda Abad.

Lastly, I would like to thank my family for all their pray, love and encouragement. The moral support of all my family member was the main stream to complete my PhD.

Thanks Malaysia.

Rashid Naseem
Software modularization plays an important role in software maintenance phase. Modularization is the breaking down of a software system into sub-systems so that most similar entities (e.g., classes or functions) are collected in clusters to get the modular architecture. To check the accuracy of collected clusters, authoritativeness is calculated which finds the correspondence between collected clusters and a software decomposition prepared by a human expert. To improve the authoritativeness, different techniques have been proposed in the literature. However, agglomerative hierarchical clusterings (AHCs) are preferred due to their resemblance with internal tree structure of the software systems because AHC results in a tree like structure, called dendrogram. AHC uses similarity measures to find association values between entities and makes clusters of similar entities. This research addresses the strengths and weakness of existing similarity measures (i.e., Jaccard (JC), JaccardNM (JNM), and Russal&Rao (RR)). For example JC measure produces large number of clusters (NoC) and number of arbitrary decisions (AD). Large NoC is considered to be better for improving the authoritativeness but large AD deteriorates it. To overcome this trade-off, new combined binary similarity measures are proposed. To further improve the authoritativeness, this research explores the idea of hierarchical clustering combination (HCC) for software modularization which is based on combining results (dendrograms) of individual AHCs (IAHCs). This research proposes an improved HCC approach in which the dendrograms are represented in a 4+N (4 is the number of features and can be extended to N) dimensional Euclidean space (4+NDES). The proposed binary similarity measures and 4+NDES based HCC approach are tested on several test software systems. Experimental results revealed [13.5% - 63.5%] improvement in authoritativeness as compared to existing approaches. Thus the combined measures and 4+NDES-HCC have shown better potential to be used for software modularization.
ABSTRAK

Modularisasi perisian memainkan peranan yang penting dalam fasa penyelenggaraan perisian. Modularisasi adalah pecahan daripada sistem perisian ke dalam sub-sistem supaya kebanyakan entiti yang sama (contohnya, kelas atau fungsi) dikumpulkan dalam kelompok tertentu untuk mendapatkan seni bina modular. Untuk memeriksa ketepatan kelompok yang telah dikumpul, keberkesanan dikira dari segi kesesuaian di antara kelompok yang telah dikumpul dan penguraian perisian yang diperincikan oleh pakar. Untuk meningkatkan keberkesanannya, pelbagai teknik yang berbeza telah dicadangkan dalam literatur. Walau bagaimanapun, agglomerative hierarchical clusterings (AHCs) lebih digemari kerana persamaan struktur dalam sistem perisian kerana AHC membentuk sebuah struktur berbentuk pokok, yang dipanggil dendrogram. AHC menggunakan penilaian persamaan untuk mencari nilai-nilai persatuan antara entiti dan membentuk kelompok entiti yang sama. Kajian ini menangani kekuatan dan kelemahan penilaian persamaan yang sedia ada (iaitu, Jaccard (JC), JaccardNM (JNM), dan Russal&Rao (RR)). Sebagai contoh, teknik penilaian JC menghasilkan sejumlah besar kelompok (NoC) dan beberapa arbitrary decisions (AD). NoC dianggap lebih baik untuk meningkatkan keberkesanan, tetapi AD besar kemungkinan merosot dari aspek tertentu. Untuk mengatasi masalah keseimbangan ini, langkah-langkah persamaan binari gabungan baru telah dicadangkan. Untuk meningkatkan lagi kewibawaan, penyelidikan ini meneroka idea hierarchical clustering combination (HCC) untuk perisian modularisasi yang berasaskan menggabungkan hasil (dendrograms) individu AHCs (IAHCs). Kajian ini mencadangkan pendekatan HCC bertambah baik di mana dendrograms diwakili dalam 4+N (4 adalah bilangan ciri-ciri dan boleh dilanjutkan kepada N) dimensi ruang Euclidean (4+NDES). Kajian ini telah melaksanakan langkah-langkah persamaan binari yang dicadangkan dan 4+NDES berdasarkan pendekatan HCC telah diuji pada beberapa sistem perisian ujian. Oleh itu 4+NDES-HCC dan langkah-langkah persamaan binari yang dicadangkan adalah lebih berpotensi untuk digunakan untuk perisian modularisasi.
# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECLARATION</td>
<td>ii</td>
</tr>
<tr>
<td>DEDICATION</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENT</td>
<td>iv</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>v</td>
</tr>
<tr>
<td>ABSTRAK</td>
<td>vi</td>
</tr>
<tr>
<td>CONTENTS</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xiv</td>
</tr>
<tr>
<td>LIST OF ALGORITHMS</td>
<td>xvi</td>
</tr>
<tr>
<td>LIST OF APPENDICES</td>
<td>xvii</td>
</tr>
<tr>
<td>LIST OF SYMBOLS AND ABBREVIATIONS</td>
<td>xviii</td>
</tr>
<tr>
<td>LIST OF PUBLICATIONS</td>
<td>xx</td>
</tr>
</tbody>
</table>

## CHAPTER 1  INTRODUCTION

1.1 Overview                                   1
1.2 Problem Statement                          5
1.3 Objectives                                 6
1.4 Scope                                      7
1.5 Dissertation Outline                       7

## CHAPTER 2  LITERATURE REVIEW

2.1 Introduction                               9
2.2 Software Maintenance                       9
2.3 Aspects of Software Modularization using Clustering 11
2.4 Application of IAHC for Software Modularization 14
2.4.1 Selection of Entities and Features       14
2.4.2 Selection of Similarity Measure           16
CHAPTER 3 RESEARCH METHODOLOGY

3.1 Introduction 52
3.2 Phase 1 52
   3.2.1 Study the Performance of Existing Similarity Measures 54
   3.2.2 Combined Binary Similarity Measures 54
   3.2.3 Comparison of Combined Binary Similarity Measures 54
       3.2.3.1 Test Systems 54
       3.2.3.2 Entities and Features 56
   3.2.4 Strategies for Comparing Combined Measures 57
   3.2.5 Selected Assessment Criteria for Phase 1 59
       3.2.5.1 External Assessment 60
       3.2.5.2 Expert Decompositions 60
       3.2.5.3 Internal Assessment 61
   3.2.6 Research Questions 62
3.3 Phase 2 62
   3.3.1 Study the Performance of the Existing HCCs 62
   3.3.2 4+N Dimensions in Euclidean Space Based HCC 63
   3.3.3 Selection of the Best Parameters 63
   3.3.4 Comparison of HCC 63
   3.3.5 Selected Evaluation Criteria for HCC 65
3.4 Chapter Summary 67
CHAPTER 4  COMBINED BINARY SIMILARITY MEASURES

4.1 Introduction
4.2 Analysis of the JC, JNM, and RR Measures
   4.2.1 JC with CL Clustering Process
   4.2.2 JNM with CL Clustering Process
   4.2.3 RR with CL Clustering Process
   4.2.4 Discussion on the Results of JC, JNM and RR Measures
4.3 Design of the Combined Binary Similarity Measures
   4.3.1 Combination of the JC and JNM Measures: “CJCJNM” Similarity Measure
   4.3.2 Analysis of CJCJNM Measure
   4.3.3 Combination of the JC and RR Measures: “CJCRR” Similarity Measure
   4.3.4 Combination of the JNM and RR Measures: “CJNMRR” Similarity Measure
   4.3.5 Addition of the JC and JNM and RR Measures: “CJCJNMRR” Similarity Measure
4.4 Chapter Summary

CHAPTER 5  EUCLIDEAN SPACE BASED HIERARCHICAL CLUSTERING COMBINATION

5.1 Introduction
5.2 4+NDES-HCC Algorithm
   5.2.1 Step 1, Initialization of the Variables
   5.2.2 Step 2, Repeat
   5.2.3 Step 3, Applying Base Clusterer (IAHC)
   5.2.4 Step 4, Dendrogram Descriptor
      5.2.4.1 (4+N) dimensions in a Euclidean Space Based Descriptor:
      5.2.4.2 BE Features
      5.2.4.3 ES Features
   5.2.5 Step 5, Loop Termination
   5.2.6 Step 6, Vectors Addition
   5.2.7 Step 7, Euclidean Distance
   5.2.8 Step 8, Recovery Technique
5.3 Illustration of (4+NDES-HCC) Algorithm
5.4 Complexity Analysis
5.5 Chapter Summary
CHAPTER 6  RESULTS AND ANALYSIS  97

6.1  Introduction  97
6.2  Assessment of the Combined Similarity Measures  97
   6.2.1  Arbitrary Decisions  97
   6.2.2  Number of Clusters  106
   6.2.3  Authoritativeness  115
   6.2.4  Overview of Results  119
   6.2.5  Comparison with COUSM  121
   6.2.6  Significance of the Authoritativeness  122
6.3  Assessment of the 4+NDES-HCC  123
   6.3.1  Number of Clusters  123
   6.3.2  Cluster-to-Cluster (c2c) Comparison  126
   6.3.3  Authoritativeness  128
   6.3.4  Significance of the Results  131
6.4  Overview of the Results for NDES  132
6.5  Chapter Summary  134

CHAPTER 7  CONCLUSIONS AND FUTURE WORKS  135

7.1  Introduction  135
7.2  Objectives Accomplished  136
7.3  Contributions  139
7.4  Directions for Future Work  140
   7.4.1  Integration of More Similarity Measures  140
   7.4.2  Integration of Partitioning Algorithms with Hierarchical Algorithms  141
   7.4.3  Application of Consensus Based Techniques  141
   7.4.4  Feature Extraction  141
   7.4.5  Description of Dendrogram  141
   7.4.6  Experiments on Other Test Systems  142
   7.4.7  Use of Other Assessment Criteria  142
   7.4.8  Big Data  142

REFERENCES  143

APPENDIX A  160

VITAE  169
LIST OF TABLES

2.1 An Example Feature Matrix 15
2.2 Similarity Matrix of Table 2.1 using the JC Measure 18
2.3 Iteration 1: Updated Similarity Matrix from Table 2.2 using the CL Method 18
2.4 Details of the Literatures 29
3.1 Details of the Test Software Systems 55
3.2 Relationships Between Classes that were used for Experiments 57
3.3 Statistics for the Indirect Features of Classes in Industrial Test Software Systems that were Used for Experiments 57
3.4 Clustering Stratagems for IAHCs 59
3.5 Personnel Statistics 61
3.6 Clusterers Stratagems for HCC 66
4.1 Iteration 2: Updated Similarity Matrix from Table 2.3 69
4.2 Iteration 3: Updated Similarity Matrix from Table 4.1 69
4.3 Iteration 4: Updated Similarity Matrix from Table 4.2 69
4.4 Iteration 5: Updated Similarity Matrix from Table 4.3 70
4.5 Iteration 6: Updated Similarity Matrix from Table 4.4 70
4.6 Iteration 7: Updated Similarity Matrix from Table 4.5 70
4.7 Similarity Matrix of Feature Matrix in Table 2.1 Using the JNM Measure 71
4.8 Iteration 1: Updated Similarity Matrix from Table 4.7 Using the CL Method 71
4.9 Iteration 2: Updated Similarity Matrix from Table 4.8 71
4.10 Iteration 3: Updated Similarity Matrix from Table 4.9 71
4.11 Iteration 4: Updated Similarity Matrix from Table 4.10 72
4.12 Iteration 5: Updated Similarity Matrix from Table 4.11 72
4.13 Iteration 6: Updated Similarity Matrix from Table 4.12 72
4.14 Iteration 7: Updated Similarity Matrix from Table 4.13 72
4.15 Similarity Matrix of Feature Matrix in Table 2.1 Using the RR Measure 73
4.16 Iteration 1: Updated Similarity Matrix from Table 4.15 Using the CL Method 73
4.17 Iteration 2: Updated Similarity Matrix from Table 4.16 73
4.18 Iteration 3: Updated Similarity Matrix from Table 4.17 73
4.19 Iteration 4: Updated Similarity Matrix from Table 4.18 74
4.20 Iteration 5: Updated Similarity Matrix from Table 4.19 74
4.21 Iteration 6: Updated Similarity Matrix from Table 4.20 74
4.22 Iteration 7: Updated Similarity Matrix from Table 4.21 74
4.23 Similarity Matrix of Feature Matrix in Table 2.1 using the CJCJNM Measure 79
4.24 Iteration 1: Updated Similarity Matrix from Table 4.23 using the CL Method 79
4.25 Iteration 2: Updated Similarity Matrix from Table 4.24 79
4.26 Iteration 3: Updated Similarity Matrix from Table 4.25 79
4.27 Iteration 4: Updated Similarity Matrix from Table 4.26 80
4.28 Iteration 5: Updated Similarity Matrix from Table 4.27 80
4.29 Iteration 6: Updated Similarity Matrix from Table 4.28 80
4.30 Iteration 7: Updated Similarity Matrix from Table 4.29 80
6.1 Experimental Results using Arbitrary Decisions for all Similarity Measures 104
6.2 Average Number of Arbitrary Decisions for all Similarity Measures 106
6.3 Experimental Results using Number of Clusters for all Similarity Measures 111
6.4 A fvc for PLC Test System 114
6.5 Average Number of Clusters for All Similarity Measures 115
6.6 MoJoFM Results for all Similarity Measures 116
6.7 Correlations of MoJoFM with Arbitrary Decisions (AD) and Number of Clusters (NoC) 117
6.8 Average MoJoFM Results for all Similarity Measures 119
6.9 Experimental Results using MoJoFM for COUSM (J-JNM) and Combined Measures 121
6.10 Statistics of $f_{cv1}$, $f_{vc2}$ and $f_{vc3}$ 122
6.11 The T-test Values between the MoJoFM Results of the Combined and Existing Measures using CL and SL Method 122
6.12 Descriptive Statistics of Results Achieved for Number of Clusters 125
6.13 Percentage Improvement of Number of Clusters for NDES over CD and IAHCs 126
6.14 Descriptive Statistics of Results Achieved for c2c 127
6.15 Percentage Improvement of c2c Values for NDES over CD and IAHCs 128
6.16 Descriptive Statistics of Results Achieved for MoJoFM 128
6.17 Percentage Improvement of MoJoFM Values for NDES over CD and IAHCs 130
6.18 The T-test Values between the MoJoFM Results of the NDES and, CD and IAHC 132
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Hierarchical clustering approaches and dendrogram</td>
</tr>
<tr>
<td>1.2</td>
<td>General framework of the HCC</td>
</tr>
<tr>
<td>2.1</td>
<td>Direct and indirect relationships in a C++ input/output libraries$^1$</td>
</tr>
<tr>
<td>2.2</td>
<td>IAHC Approach</td>
</tr>
<tr>
<td>2.3</td>
<td>Dendrogram</td>
</tr>
<tr>
<td>2.4</td>
<td>General HCC approach</td>
</tr>
<tr>
<td>3.1</td>
<td>Proposed research framework</td>
</tr>
<tr>
<td>3.2</td>
<td>Improved IAHC</td>
</tr>
<tr>
<td>3.3</td>
<td>IAHCs with combined similarity measures and linkage methods</td>
</tr>
<tr>
<td>3.4</td>
<td>Proposed 4+NDES-HCC approach</td>
</tr>
<tr>
<td>4.1</td>
<td>Number of clusters and arbitrary decisions created by JC and JNM</td>
</tr>
<tr>
<td>5.1</td>
<td>The 4+N dimensions in euclidean space based HCC model</td>
</tr>
<tr>
<td>5.2</td>
<td>Different steps of the 4+NDES-HCC algorithms. Two dendrograms ($D^1$ and $D^2$) are described using 4+NDES and then V-Matrices are aggregated into AV-Matrix. Lastly a recovery tool is used to get the final hierarchy $FH$</td>
</tr>
<tr>
<td>6.1</td>
<td>Experimental results of arbitrary decisions for all binary measures using IAHCs</td>
</tr>
<tr>
<td>6.1</td>
<td>Experimental results of arbitrary decisions for all binary measures using IAHCs (Continued)</td>
</tr>
<tr>
<td>6.1</td>
<td>Experimental results of arbitrary decisions for all binary measures using IAHCs (Continued)</td>
</tr>
<tr>
<td>6.2</td>
<td>Arbitrary decisions taken during clustering process by all binary similarity measures using CL and SL methods for different test software systems.</td>
</tr>
<tr>
<td>6.3</td>
<td>Experimental results for the number of clusters</td>
</tr>
<tr>
<td>6.3</td>
<td>Experimental results for the number of clusters (continued)</td>
</tr>
<tr>
<td>6.3</td>
<td>Experimental results for the number of clusters (continued)</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>6.4</td>
<td>Number of clusters created by all binary similarity measures during clustering process using CL and SL methods for different test software systems.</td>
</tr>
<tr>
<td>6.5</td>
<td>Size of the clusters created using DDA test system and CL method</td>
</tr>
<tr>
<td>6.6</td>
<td>Results for number of clusters created by different stratagems</td>
</tr>
<tr>
<td>6.7</td>
<td>Number of clusters created by different stratagems in each iteration</td>
</tr>
<tr>
<td>6.8</td>
<td>Results for c2c created by different stratagems</td>
</tr>
<tr>
<td>6.9</td>
<td>c2c results for each iteration created by different stratagems</td>
</tr>
<tr>
<td>6.10</td>
<td>Results for authoritativeness created by different stratagems</td>
</tr>
<tr>
<td>6.11</td>
<td>Authoritativeness results for each iteration created by different stratagems</td>
</tr>
<tr>
<td>6.12</td>
<td>Percentage improvement of NDES with respect to existing measures (JC, JNM, and RR) using CL, SL, and WL methods</td>
</tr>
</tbody>
</table>
## LIST OF ALGORITHMS

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Individual Agglomerative Hierarchical Clustering (IAHC) Algorithm</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>Complete Linkage (CL) Method</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>Single Linkage (SL) Method</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>Weighted Average Linkage (WL) Method</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>4+N Dimensions in Euclidean Space based HCC</td>
<td>89</td>
</tr>
</tbody>
</table>
LIST OF APPENDICES

A  Propositions  160
# LIST OF SYMBOLS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AHC</td>
<td>Agglomerative Hierarchical Clustering</td>
</tr>
<tr>
<td>ACDC</td>
<td>Algorithm for Comprehension Driven Clustering</td>
</tr>
<tr>
<td>c2c</td>
<td>Cluster to Cluster</td>
</tr>
<tr>
<td>CLH</td>
<td>Cannot Link Hard</td>
</tr>
<tr>
<td>CH</td>
<td>Cluster Cohesion</td>
</tr>
<tr>
<td>CC/G</td>
<td>Graph-based Clustering Approach</td>
</tr>
<tr>
<td>CPCC</td>
<td>Co-Phenetic Correlation Co-efficiency</td>
</tr>
<tr>
<td>CL</td>
<td>Complete Linkage</td>
</tr>
<tr>
<td>COUSM</td>
<td>Cooperative Only Update Similarity Matrix</td>
</tr>
<tr>
<td>DDA</td>
<td>Document Designer Application</td>
</tr>
<tr>
<td>DSM</td>
<td>Dependency Structure Matrix</td>
</tr>
<tr>
<td>dsm</td>
<td>design structure matrix clustering</td>
</tr>
<tr>
<td>EdgeSim</td>
<td>Edge Similarity</td>
</tr>
<tr>
<td>eb</td>
<td>edge betweenness clustering</td>
</tr>
<tr>
<td>ECA</td>
<td>equal size cluster approach</td>
</tr>
<tr>
<td>FCA</td>
<td>Formal Concept Analysis</td>
</tr>
<tr>
<td>FES</td>
<td>Fact Extractor System</td>
</tr>
<tr>
<td>FACA</td>
<td>Fast Community Algorithm</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>HCC</td>
<td>Hierarchical Clusterers Combinations</td>
</tr>
<tr>
<td>IAHC</td>
<td>Individual Agglomerative Hierarchical Clustering</td>
</tr>
<tr>
<td>IEEE</td>
<td>The Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IL</td>
<td>Information Loss</td>
</tr>
<tr>
<td>J-JNM</td>
<td>Jaccard and JaccardNM</td>
</tr>
<tr>
<td>JC</td>
<td>Jaccard</td>
</tr>
<tr>
<td>JNM</td>
<td>JaccardNM</td>
</tr>
<tr>
<td>LIMBO</td>
<td>Scalable Information Bottleneck</td>
</tr>
<tr>
<td>LSI</td>
<td>Latent Semantic Indexing</td>
</tr>
<tr>
<td>Max</td>
<td>Maximum</td>
</tr>
<tr>
<td>MQ/mq</td>
<td>Modularization Quality</td>
</tr>
<tr>
<td>MLH</td>
<td>Must Link Hard</td>
</tr>
<tr>
<td>MCA</td>
<td>Maximizing Cluster Approach</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>MST</td>
<td>Minimum Spanning Tree</td>
</tr>
<tr>
<td>MMST</td>
<td>Modified Minimum Spanning Tree</td>
</tr>
<tr>
<td>MoJoFM</td>
<td>Move and Join Effectiveness Measure</td>
</tr>
<tr>
<td>MoJo</td>
<td>Move and Join</td>
</tr>
<tr>
<td>MED</td>
<td>Maximum Edge Distance</td>
</tr>
<tr>
<td>MeCl</td>
<td>Merge Clusters</td>
</tr>
<tr>
<td>MATCH</td>
<td>Min-transitive Combination of Hierarchical Clusterings</td>
</tr>
<tr>
<td>NAHC</td>
<td>Nearest hill-climbing</td>
</tr>
<tr>
<td>NoC</td>
<td>Number of Clusters</td>
</tr>
<tr>
<td>NDES</td>
<td>N Dimensional Euclidean Space</td>
</tr>
<tr>
<td>OAM</td>
<td>Optimal Algorithm for MoJo</td>
</tr>
<tr>
<td>PEDS</td>
<td>Power Economic Dispatch System</td>
</tr>
<tr>
<td>PLC</td>
<td>Printer Language Converter</td>
</tr>
<tr>
<td>PLP</td>
<td>Print Language Parser</td>
</tr>
<tr>
<td>RR</td>
<td>Russal&amp;Rao</td>
</tr>
<tr>
<td>SAVT</td>
<td>Statistical Analysis Visualization Tool</td>
</tr>
<tr>
<td>SBSE</td>
<td>Search Based Software Engineering</td>
</tr>
<tr>
<td>SBSC</td>
<td>Search Based Software Clustering</td>
</tr>
<tr>
<td>SL</td>
<td>Single Linkage</td>
</tr>
<tr>
<td>SAHC</td>
<td>Steepest-Ascend Hill-Climbing</td>
</tr>
<tr>
<td>SD</td>
<td>Sorenson-Dice</td>
</tr>
<tr>
<td>std.dev.</td>
<td>Standard Deviation</td>
</tr>
<tr>
<td>sig.</td>
<td>Significance</td>
</tr>
<tr>
<td>TF.IDF</td>
<td>Term Frequency Inverse Document Frequency</td>
</tr>
<tr>
<td>WCA</td>
<td>Weighted Combined Algorithm</td>
</tr>
<tr>
<td>WL</td>
<td>Weighted Average Linkage</td>
</tr>
</tbody>
</table>
LIST OF PUBLICATIONS

- Rashid Naseem, Mustafa Mat Deris, Onaiza Maqbool, Jing-peng Li, Sara Shahzad, and Habib Shah (2016), Improved binary similarity measures for software modularization, *Frontiers of Information Technology & Electronic Engineering (FITEE)*, In press (ISI and Scopus Indexed, Springer Journal)

- Rashid Naseem, Mustafa Mat Deris and Onaiza Maqbool (2014), Software Modularization Using Combination of Multiple Clustering, *IEEE 17th International Multi-Topic Conference (INMIC)*, pp. 277 - 281, IEEE Conference

CHAPTER 1

INTRODUCTION

1.1 Overview

The software development life cycle comprises of many phases, perhaps the most important phase being maintenance, because it increases the operational life of a software after it has been deployed. Software maintenance is considered to be a difficult phase since it dominates other phases in terms of cost and efforts required (Garcia et al., 2011). Bavota et al. (2013) stated that a typical system’s maintenance costs up to 90% of the total project expenses. In addition to that, Kumari et al. (2013) reported that maintenance phase requires 50% of the human and computer resources. According to Antonellis et al. (2009), United States spends $60 billion per year of its economy on the maintenance of software systems. Keeping in view the above, software maintenance may be considered a competitive research area to reduce the efforts required for maintenance.

In the software maintenance phase, “understanding the architecture” of software systems is an important and challenging activity to start adding new requirements or to start reverse engineering (Muhammad et al., 2012). However, when new requirements are incorporated in software systems, the systems’ size and complexity increases (Shtern & Tzerpos, 2014), while architecture may deteriorate and diverge from the original documentation (Shtern & Tzerpos, 2014; Chardigny et al., 2008) due to the many reasons: 1) software development without an architecture design phase; 2) original developer may not be available; 3) documentation may not be uptodated and; 4) copied source code without understanding the code segments.
Therefore, it becomes difficult to understand the systems and introduce changes to them. Corazza et al. (2011), Cornelissen et al. (2009) and Kuhn et al. (2007) reported that “understanding” the systems costs up to 60% of the whole maintenance cost. Hence, researchers have explored the problem and are working to solve it using automated techniques for the last two decades. These techniques support the software maintenance team in terms of gathering and presenting the basic architectural information for better understanding and improvement of the software systems. Architectural information comprises of a software architecture (structures of the systems, e.g., module architecture), which plays a vital role to understand the software systems (Ducasse & Pollet, 2009). Therefore, a number of approaches have been developed and proposed in the literature to recover the architecture of software systems mainly from their source code in order to improve the authoritativeness. Authoritativeness determines how much the automatically obtained decomposition is similar to manual decomposition prepared by human expert (Wu et al., 2005). Besides hierarchical clustering (Maqbool & Babri, 2007b; Cui & Chae, 2011; Andritsos & Tzerpos, 2005), these approaches include, supervised clustering (Hall et al., 2012), optimization techniques (Praditwong et al., 2011), role based recovery (Dugerdil & Jossi, 2009), graph based techniques (Bittencourt & Guerrero, 2009), association based approaches (Vasconcelos & Werner, 2007), spectral method (Xanthos & Goodwin, 2006), rough set theory (Jahnke, 2004), concept analysis (Tonella, 2001), and visualization tools (Synytskyy et al., 2005).

Clustering is an emerging research area to acquire different types of knowledge from the data by forming meaningful clusters (groups). Thus, entities within a cluster have similar characteristics or features, and are dissimilar from entities in other clusters. To determine similarity based on features of an entity, a similarity measure is employed. Finding a good clustering is an NP-complete problem (Tumer & Agogino, 2008). To address the problem, a number of clustering algorithms exist. Moreover algorithms have been proposed for various domains, e.g., DNA analysis (Avogadri & Valentini, 2009), software modularization (Wang et al., 2010), bioinformatics (Janssens et al., 2007), image processing (Gong et al., 2013) and information retrieval
Each clustering algorithm produces results according to some criteria and bias of the technique (Faceli et al., 2007). For example, Complete linkage clustering algorithm creates small size of clusters because this algorithm considers maximum distance between clusters while Single linkage algorithm considers minimum distance between the clusters and therefore makes large size clusters.

Clustering techniques can be mainly divided into two categories: 1) partitional and 2) hierarchical. Partitional clustering makes flat partitions (or clusters) in the input data, while hierarchical clustering results in a tree like nested structure of clusters called “dendrogram”. The two main types of hierarchical clustering are divisive and agglomerative. The divisive approach starts by considering the whole data as one big cluster and then iteratively splits the clusters into two nested clusters in a top-down manner. Agglomerative hierarchical clustering (AHC) starts with singleton clusters and merges two most similar clusters at every step in a bottom-up manner as shown in Figure 1.1.

Figure 1.1: Hierarchical clustering approaches and dendrogram

Due to the ill-posed problem of clustering (Kashef & Kamel, 2010; Ghaemi et al., 2009), different clustering algorithms usually produce different results from the given data. For example, if a clustering algorithm produces results which are stable (results are stable if affect of slight modification in input is also slight on output
of algorithm), it may produce less authoritative results and vice versa. To improve clustering results many of the existing methods have been modified and improved (Chen et al., 2014; Wang & Su, 2011; Forestier et al., 2010; Kashef & Kamel, 2010). Some efforts have been made to improve the results by using prior information, e.g., user input and number of clusters, but this information is very difficult to obtain in advance (Ghaemi et al., 2009).

Recently, combining the individual AHC (IAHC) algorithms in an ensemble fashion has gained the attention of the researchers to boost the clustering accuracy, known as Hierarchical Clustering Combination (HCC). This ensemble clustering is achieved by combining the dendrograms created by different IAHCs, to obtain a consensus result (Rashedi et al., 2015; Zheng et al., 2014). Generally this combination has four phases as shown in Figure 1.2. In the first phase the IAHCs are applied to the input data to create dendrograms. Then the dendrograms are translated into an intermediate format (mostly in a matrix format called Description Matrix), so that they can be aggregated into a single intermediate format. Lastly a recovery tool is applied to recover a single consensus integrated dendrogram for the input data.

![Figure 1.2: General framework of the HCC](image)

For software modularization AHC algorithms have been widely used by researchers to cluster the software systems in order to improve the authoritativeness (Muhammad et al., 2012; Shtern & Tzerpos, 2010; Patel et al., 2009; Maqbool & Babri, 2007b; Mitchell, 2006). However, the similarity measures used by AHCs perform better for one assessment criteria and poor for another. To overcome this trade off, integration of the existing similarity measures are performed. Moreover, HCC which has never been explored for software modularization is introduced. In addition to that, the existing HCC approach has the limitation of describing dendrogram using only
a single feature and considering that feature as a distance value to make the clusters
for final dendrogram. Therefore, in this research, a new improved HCC approach is
proposed which is based on Euclidean space theory and is named as 4+NDES-HCC.
This approach translate the dendrogram using four new features as vector dimensions
in Euclidean space. The entity points in dendrograms are represented as vectors. Then,
the corresponding vectors of two dendrograms are added using vector addition property
to get the aggregated vector matrix. Then, Euclidean distance measure is applied to get
the distance matrix. At the end a recovery tool is used such as IAHC to get the final
consensus dendrogram which provides high authoritativeness.

1.2 Problem Statement

HCC has gain the attention of the researcher due to its promising results (Rashedi
et al., 2015; Zheng et al., 2014; Rashedi & Mirzaei, 2013; Ghosh & Acharya,
2011). HCC bases on the results (dendrogram) of IAHCs while IAHC bases on
a linkage method and a similarity measure. However, similarity measures have a
major influence on the clustering results as compared to a linkage method (Shtern
& Tzerpos, 2012). For software modularization comparative studies reported that
Jaccard (JC), Jaccard-New-Measure(JNM), and Russal&Rao (RR) binary similarity
measures produced better clustering results as compared to Euclidean, Simple and
Rogers&Tanimoto measures (Naseem et al., 2013; Shtern & Tzerpos, 2012; Cui &
Chae, 2011). These similarity measures have different characteristics, for example,
JC creates large number of clusters with large number arbitrary decisions while JNM
reduces the arbitrary decisions but at the cost of reducing number of clusters. Similarly
RR has the strengths to reduce the arbitrary decisions where JC creates. Producing
large number of clusters and reducing the arbitrary decisions ensue into better and
qualitative software modularization results (Naseem et al., 2013; Shtern & Tzerpos,
2012; Cui & Chae, 2011). This research is motivated by the idea of “integrating the
strengths of these measures (i.e., large number of clusters while taking less arbitrary
decisions) in a single binary similarity measures to improve the clustering quality”.
Besides the aforementioned problem, this research also investigates the existing HCC approaches. As stated that HCC combines the dendrograms to create a single consensus dendrogram. Therefore, the dendrograms are translated into an intermediate format (mostly in a matrix format called Description Matrix), so that they can be aggregated into a single intermediate format for the recovery of final dendrogram. This research is also inspired by the problem that “descriptor matrices described by previous researchers (Rashedi et al., 2015; Mirzaei et al., 2008), allows to extract and utilize only the distances between two entities on the bases of a single type of relationship present between any two entity points in the dendrogram”. For example, CD descriptor (Mirzaei et al., 2008) uses the height level in a dendrogram as a distance (value) between two entity points. Hence this research came up with the idea to explore and extract features (dimensions) from the dendrograms to improve the clustering quality and introduce new HCC approach based on these features, namely 4+NDES-HCC.

1.3 Objectives

Based on the problem statement, following objectives are formulated:

1. To propose combined binary similarity measures that integrate the strengths of existing binary similarity measures thus reducing arbitrary decisions and increase the number of clusters which lead the IAHCs to improve the clustering quality in terms of authoritativeness, for software modularization;

2. To propose an improved hierarchical clustering combination approach based on Euclidean space and distance measure, namely 4+NDES-HHC 4 presents extracted number of features (dimensions) which can be extended to any number (N) using Euclidean space;

3. To evaluate the performance of combined similarity measures proposed in Objective 1 and 4+NDES-HCC clustering approach proposed in Objective 2 using the arbitrary decisions, number of clusters, cluster to cluster and authoritativeness, and to compare these with existing clustering approaches such as cooperative clustering, CD-HCC, and IAHC-CL.
1.4 Scope

This research only focuses on software modularization through agglomerative hierarchical clustering approaches. This research analyzes the proposed and existing clustering approaches using open source (Weka, Mozilla, Junit, and iText) and proprietary industrial (DDA, FES, PEDS, PLC, PLP, and SAVT) software systems. To evaluate experimental results, MoJoFM (Wen & Tzerpos, 2004a) for authoritativeness and cluster to cluster are used as the external assessment criteria and for internal assessment, arbitrary decisions and number of clusters created by the clustering approach are used.

1.5 Dissertation Outline

This chapter gives the overview with motivation for software modularization using clustering techniques, problem statement and research objectives with the scope of this research. The remaining chapters are organized as follows:

- Chapter 2 provides an overview of clustering approach for software modularization and a survey of literature. This chapter discusses the individual clustering algorithms used for software modularization and hierarchical clusterers combinations techniques. The review of comparative published literature and evaluation criteria is also discussed in this chapter.

- Chapter 3 presents the research methodology. In this chapter, a research framework is proposed which shows how the research has been conducted. The proposed framework comprised of two phase. First phase presents the steps for analysis of existing binary similarity measures and introducing the new combined similarity measures. Second phase proposes the new HCC based approach, i.e., 4+NDES-HCC. Moreover, this chapter also explains the experimental setup.

- Chapter 4 analyzes the existing well known binary similarity measures for software modularization in detail to identify their strengths and weakness.
Based on these strengths, this chapter proposes the combined binary similarity measures and discusses a small case study where the proposed measures perform better than the existing measures.

- **Chapter 5** presents the Euclidean space based HCC approach, named 4+NDES-HCC. To overcome the limitation of existing HCC approach, 4+NDES-HCC extracts different features using Euclidean space from the dendrogram created by IAHCs/clusterers and then combine the features using vector addition property of Euclidean space. The distance between entities is calculated using Euclidean distance measure. To produce final consensus dendrogram an IAHC is used.

- **Chapter 6** presents and discusses the experimental results of combined and existing binary similarity measures, and also presents the results for 4+NDES-HCC approach. The results for similarity measures are analyzed by formulating 19 research questions (as given in Chapter 3). Moreover, the results for 4+NDES-HCC are compared with CD-HCC and IAHC using evaluation criteria, e.g., number of clusters, cluster to cluster, and authoritativeness. To get insight of the results, this chapter gives the t-test to find the significance of the results of the proposed combined similarity measures and 4+NDES-HCC clustering approach.

- **Chapter 7** concludes this dissertation with achieved objectives and contributions. To enhance the current research this chapter gives a list of future directions.
CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

As discussed in the previous chapter, clustering techniques are widely used for the software modularization. Likewise, this chapter reviews the related works to the software modularization, clustering, HCC, and evaluation criteria used for analysis of the modularization results.

This chapter is organized as follows. Section 2.2 provides an overview of the software maintenance. Section 2.3 gives an overview of commonly used aspects of the software modularization using IAHCs. The overview of software modularization using IAHC is given in Section 2.4. Section 2.5 surveys the related literature of software modularization using IAHCs while Section 2.6 presents the survey of Non-IAHCs’ literature. Researchers have compared different techniques for software modularization, summary of these comparative literature is given in Section 2.7. Section 2.8 outlines the survey of existing HCC works. Works related to the evaluation of the software modularization results are presented in Section 2.9. The overall scenario which leads to over proposed research methodology is discussed in Section 2.10. Finally, this chapter is concluded by Section 2.11.

2.2 Software Maintenance

According to IEEE Std 14764-2006, software maintenance is the “modification of a software product after delivery to correct faults, to improve performance or
other attributes, or to adapt the product to a changed environment”. Knowledge of application domain, system’s architecture, algorithms used, past and new requirements, and experience on the execution of a particular software process is required for maintenance of the software systems (Chikofsky & Cross, 1990).

According to Huselius (2007), industries are facing major challenges of improving and maintaining the software systems due to the high expensive cost. It is estimated that 70% of the cost is spent on maintenance after softwares first release (Belle, 2004). Major changes are made in response to user requirements, data formats and hardware. According to Waters (2004), most researchers agree that 40 to 80 percent of software activities are focused on maintenance and evolution. However a recent study reported that a typical system’s maintenance costs up to 90% of the total project expenses (Bavota et al., 2013). Due to the high cost of maintenance, researchers devoted their efforts to reduce the maintenance’s cost by introducing automated techniques. These techniques support the software maintenance regarding assembling and introducing the essential architectural information for better understanding, change and improvement of the software systems. Architectural information comprises of a software architecture (structures of the systems, e.g. module architecture), which plays a fundamental part to comprehend the software systems (Ducasse & Pollet, 2009).

Garlan (2000) uncovered the significance of software architecture by anticipating its part on six unique viewpoints: understanding, construction, reuse, management, evolution, and analysis. Adding to that, Lutellier et al. (2015) argued that architecture is important for programmer communication, program comprehension, and software maintenance. In any case, for some software systems, the representation of their architecture is not available or updated, because of the many reasons. This phenomena is known as architectural drift and erosion (Medvidovic & Taylor, 2010). These marvels are brought on via unintended, careless addition, removal, and modification of architectural design decisions. To manage drift and erosion, at some point or another researchers are compelled to recuperate architecture from the available resources of the software systems. Therefore, a number of automatic approaches have been introduced to cluster the software systems for architecture recovery. The
software clustering can also be utilized for the accompanying purposes: 1) to improve the structure of software; 2) to present the higher level of abstraction of the software system; 3) to facilitate in presenting the different level of architectural views and 4) to arrange, outline and execute the changes in the source code furthermore predicts the effects of these predicts in the source code.

2.3 Aspects of Software Modularization using Clustering

Software clustering has three major aspects (Shtern & Tzerpos, 2012). First aspect is the software entities to cluster. These software entities can be variables, subprograms and files in case of structured systems (Koschke & Eisenbarth, 2000), while in case of object oriented systems classes can be considered as entities to cluster (Scanniello et al., 2010a; Cai et al., 2009; Sindhgatta & Pooloth, 2007). In recent days the popularity of components based software development has introduced components or subsystems as entities because these subsystems are clustered into larger subsystems (Mitchell et al., 2002). Clustering entities also decide the level of clustering. If clustering entities are variables and subprograms, these can be combined into small clusters equivalent to classes and objects (Koschke & Eisenbarth, 2000). However, if clustering entities are classes and files then outcome of clustering process will be subsystems.

Second important aspect is similarity measures which decide the degree of closeness between two software entities. These similarity measures calculate how much two entities compact are. This association of compactness between entities is calculated using the features/relationships between entities. These features can be obtained through static analysis or dynamic analysis of source code. In static analysis, these features are obtained without executing software system. So these are obtained directly from source code syntax. However in case of dynamic analysis, software is executed and features are captured from execution traces. A recent study shows that static features perform better as compared to dynamic features for software modularization (Sajnani & Lopes, 2014). In structured systems indirect
features are mostly function call, data accesses and code distribution in files (Koschke & Eisenbarth, 2000). A very good list of these relationships for object oriented systems is compiled by Bauer & Trifu (2004). This list includes inheritance coupling, aggregation coupling, association coupling, access coupling and indirect coupling. This list is extended by Muhammad et al. (2012) and extracted 26 features. Authors considered sixteen features as direct features and ten indirect features. A direct relationship between two entities (in this case classes) is known as a direct feature, for example, inheritance. Indirect is the common relationship that exists between two entities, for example, two classes accessing a library. To illustrate the direct and indirect relationship, consider the case in Figure 2.1 which demonstrates an example software system with various relations between classes. This example has taken from the online\(^1\) C++ reference documentation. All the classes available in the figure have inheritance relationships. Circle presents that all the entities have direct inheritance relationship while rectangles comprising entities show that these entities share/accessing a common entity. If direct inheritance relationships were to be considered, basic_ios, basic_ostream and basic_ostringstream could be placed in one group. Despite the fact that this is a valuable view, another possibility is, basic_ostringstream and basic_ofstream have a common indirect relationship, i.e., accessing basic_ostream, therefore could be placed in one group which presents an alternate however significant view of the system. Muhammad et al. (2012) reported that for software modularization indirect features produce better results.

Features may be in the binary or non-binary form. A binary feature represents the presence or absence of a relationship between two entities, while non-binary features are weighted features using different weighting schemes, for example, absolute and relative (Cui & Chae, 2011), to demonstrate the strength of the relationship between entities. Binary features are widely used in software modularization (Muhammad et al., 2012; Cui & Chae, 2011; Mitchell & Mancoridis, 2006; Wiggerts, 1997).

Third concern for software clustering is techniques employed to perform

\(^1\)http://naipc.uchicago.edu/2015/ref/cppreference/en/cpp/io.html
clustering process. These techniques are in the form of algorithms and have various forms. Clustering techniques can be categorized as hierarchical clustering, optimization clustering, graph theoretical, and partitional clustering. Hierarchical clustering can take one of the two forms, divisive or agglomerative (Maqbool, 2006). In case of divisive algorithms, all entities are placed in one cluster initially and then partition process is performed until desired number of clusters is obtained. However, agglomerative clustering (i.e. IAHC) takes reverse approach and places all entities in their own clusters. So initially the number of entities is equal to the number of clusters. Then clusters are combined on the bases of similarity measures until the desired number of clusters is obtained. Optimization clustering algorithms use optimization techniques to maximize intra clustering similarity and minimize the inter cluster similarity. In case of graph theoretical algorithms, entities and their relationships are represented as graphs with nodes representing entities and edges as relationships.

Partitional clustering produces flat clusters with no hierarchy, and requires prior knowledge of the number of clusters. In the software domain, partitional clustering has also been used (Shah et al., 2013; Kanellopoulos et al., 2007; Lakhota, 1997),
however there are some advantages of using IAHC. For example, IAHC does not require prior information about the number of clusters. Moreover, Wiggerts (1997) stated that the process of IAHC is very similar to the approach of reverse engineering where architecture of a software system is recovered in a bottom-up fashion. IAHC provides different levels of abstraction and can be useful for the end user to select the desired number of clusters when the modularization results are meaningful to him (Lutellier et al., 2015). Since a maintainer may not have knowledge of the number of clusters in advance, therefore viewing the architecture at different abstraction levels facilitates understanding. Techniques have also been proposed to select an appropriate abstraction level, e.g., Chong et al. (2013) proposed a dendrogram cutting approach for this purpose.

2.4 Application of IAHC for Software Modularization

When IAHC is utilized for software modularization, the first step that occurs is the selection of entities to be clustered where each entity is described by different features. The steps are presented in more detail in the following subsections.

2.4.1 Selection of Entities and Features

Selecting the entities and features associated with entities depends on the type of software system and the desired architecture (e.g., layered/module architectures) to be recovered. For software modularization, researchers have used different types of entities, e.g. methods (Saeed et al., 2003), classes (Bauer & Trifu, 2004), and files (Andritsos & Tzerpos, 2005). Researchers have also used different types of features to describe the entities such as global variables used by an entity (Muhammad et al., 2012), and procedure calls (Andritsos & Tzerpos, 2005). Features may be in binary or non-binary format. A binary feature represents the presence or absence of a relationship between two entities, while a non-binary feature is a weighted feature, which demonstrates the strength of relationship between entities. However, binary features are widely used in software modularization, because they give better results
as compared to other types (Naseem et al., 2013; Cui & Chae, 2011; Mitchell & Mancoridis, 2006).

To apply IAHC, a software system must be parsed to extract the selected entities and features associated with entities. This process results in a feature matrix of size \( N \times P \), where \( N \) is the total number of entities and \( P \) is the total number of features. Each entity in the feature matrix has a feature vector \( f_i = \{f_1, f_2, f_3, ..., f_P\} \). More generally, \( F \) presents a general feature matrix which takes values from \( \{0,1\}^p \), in other words \( F = \{0,1\}^p \), where ‘1’ means presence of a feature and ‘0’ otherwise. IAHC takes \( F \) as input, as shown in Algorithm 1 (Figure 2.2 shows the process model). Table 2.1 shows an example 0-1 feature matrix \( F \) of a very small example software system, which contains 8 entities (E1 - E8) and 13 features (f1 - f13).

Table 2.1: An Example Feature Matrix

<table>
<thead>
<tr>
<th></th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
<th>f5</th>
<th>f6</th>
<th>f7</th>
<th>f8</th>
<th>f9</th>
<th>f10</th>
<th>f11</th>
<th>f12</th>
<th>f13</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Algorithm 1: Individual Agglomerative Hierarchical Clustering (IAHC) Algorithm

Input: Feature \( (F) \) matrix
Output: Hierarchy of Clusters (Dendrogram)

1: Create a similarity matrix by calculating similarity using a **Similarity Measure** between each pair of entities
2: repeat
3: Group the most similar (singleton) clusters into one cluster (using maximum value of similarity in similarity matrix)
4: Update the similarity matrix by recalculating similarity using a **Linkage Method** between newly formed cluster and existing (singleton) clusters
5: until the required number of clusters or a single large cluster is formed
2.4.2 Selection of Similarity Measure

The first step of the IAHC process is to calculate the similarity between each pair of entities to obtain a similarity matrix by using a similarity measure, as shown in Step 1 of Algorithm 1. The well known binary similarity measures for software modularization (Cui & Chae, 2011; Naseem et al., 2013), are based on the following Definition 1:

**Definition 1.** Let $E_i$ and $E_j$ be two entities and $(E_i, E_j)$ show a pair of entities, $\forall E_i, E_j$
∈ F (F represents feature matrix, see Section 2.4.1). Let a be the number of features common to both entities or pair of entities, b be the number of features present in Ei, but not in Ej, c be the number of features present in Ej, but not in Ei, d be the number of features absent in both entities (Lesot & Rifqi, 2009).

It is important to note that a+b+c+d is a constant value and is equal to the total number of features P. a + b = 0 only occurs when Ei has the feature vector fi = (0, ...,0). Likewise, a+c=0 shows that Ej has feature vector fj = (0, ...,0).

The binary similarity measures based on Definition 1, are as follows:

\[ Jaccard(JC) = \frac{a}{a + b + c} \]  (2.1)

\[ JaccardNM(JNM) = \frac{a}{2(a + b + c) + d} \]  (2.2)

\[ Russell&Rao(RR) = \frac{a}{a + b + c + d} \]  (2.3)

where JC considers only a, b, and c between entities. JNM considers all four quantities associated with the pair of entities (i.e., a, b, c, and d) with double value of a, b, and c in denominator. While RR divides a by total number of features.

**Definition 2.** A binary similarity measure, say SM, is a function whose domain is \( \{0,1\}^p \), and whose range is \( \mathbb{R}^+ \), i.e. \( SM : \{0,1\}^p \rightarrow \mathbb{R}^+ \) (Veal, 2011), with the following properties:

- **Positivity:** \( SM(Ei,Ej) \geq 0 \), \( \forall Ei, Ej \in F \)
- **Symmetry:** \( SM(Ei,Ej) = SM(Ej,Ei) \), \( \forall Ei, Ej \in F \)
- **Maximality:** \( SM(Ei,Ei) \geq SM(Ei,Ej) \), \( \forall Ei, Ej \in F \)

To illustrate the calculation, for instance, of the JC measure as defined in Equation 2.1, Table 2.2 gives the similarity matrix of the feature matrix shown in Table 2.1. The similarity between E1 and E2 is calculated using the quantities defined by a, b, c and d, and in this case a = 8, b = 0, c = 0, and d = 5. Putting all these values in JC similarity measure, similarity value ‘1’ (shown in Table 2.2) is obtained. Likewise, similarity values are calculated for each pair of entities.
In the first iteration of IAHC, Step 2 of Algorithm 1 searches for a maximum similarity value in Table 2.2 but it finds maximum similarity value ‘1’ which appears two times. Hence, there are two arbitrary decisions as (E1E2) has similarity value equal to 1, meanwhile (E3E4) also has the same similarity value. At this stage, IAHC may select either, but IAHC is explicitly forced to select the last value, i.e., similarity value of (E3E4), so (E3E4) cluster is made (see Table 2.3).

Table 2.2: Similarity Matrix of Table 2.1 using the JC Measure

<table>
<thead>
<tr>
<th></th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E4</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E5</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0.4</td>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E6</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E7</td>
<td>0.363</td>
<td>0.363</td>
<td>0</td>
<td>0</td>
<td>0.333</td>
<td>0.222</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E8</td>
<td>0.363</td>
<td>0.363</td>
<td>0</td>
<td>0</td>
<td>0.333</td>
<td>0.222</td>
<td>0.4</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: Iteration 1: Updated Similarity Matrix from Table 2.2 using the CL Method

<table>
<thead>
<tr>
<th></th>
<th>E1</th>
<th>E2</th>
<th>(E3E4)</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E3E4)</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E5</td>
<td></td>
<td></td>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E6</td>
<td></td>
<td></td>
<td>0.5</td>
<td>0.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E7</td>
<td>0.363</td>
<td>0.363</td>
<td>0</td>
<td>0.333</td>
<td>0.222</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E8</td>
<td>0.363</td>
<td>0.363</td>
<td>0</td>
<td>0.333</td>
<td>0.222</td>
<td>0.4</td>
<td></td>
</tr>
</tbody>
</table>

2.4.3 Selection of the Linkage Method

When a new cluster is formed, the similarities between new and the existing clusters are updated using a linkage method, as shown in Step 3 of Algorithm 1. This study applies the following well established linkage methods (see Algorithms 2, 3, and 4), which are widely used for software modularization.
Algorithm 2 - Complete Linkage (CL) Method

**Input:** Similarity Matrix  
**Output:** Updated Similarity Matrix

1: Update similarity matrix by calculating similarity between newly formed cluster and the existing entities or clusters. Let suppose three entities $E_i$, $E_j$ and $E_k$ and the newly formed cluster is $E_iE_j$. Now similarity between $E_k$ and newly formed cluster $E_iE_j$ is calculated as:

$$CL(E_iE_j, E_k) = \min(SM(E_i, E_k), SM(E_j, E_k))$$

Algorithm 3 - Single Linkage (SL) Method

**Input:** Similarity Matrix  
**Output:** Updated Similarity Matrix

1: Update similarity matrix by calculating similarity between newly formed cluster and the existing entities or clusters. Let suppose three entities $E_i$, $E_j$ and $E_k$ and the newly formed cluster is $E_iE_j$. Now similarity between $E_k$ and newly formed cluster $E_iE_j$ is calculated as:

$$SL(E_iE_j, E_k) = \max(SM(E_i, E_k), SM(E_j, E_k))$$

In the illustrative example, the similarity values are updated between a new cluster ($E_3E_4$) and existing singleton clusters using CL method as shown in Table 2.3. For example, the CL method returns the minimum value between the similarity value of $E_3$ and $E_1$ (0) and the similarity value of $E_4$ and $E_1$ (0). Both of the returned values are the same (if there was a minimum, that would be selected), therefore, IAHC selects this similarity value as the new similarity between ($E_3E_4$) and $E_1$. Similarly, all similarity values are updated between ($E_3E_4$) and the remaining entities.

IAHC repeats Steps 2 and 3 until all entities are merged in one large cluster, or the desired number of clusters is obtained. At the end IAHC results in a hierarchy of clusters, also known as dendrogram, which is shown for the current example in Figure 2.3. The obtained hierarchy is then evaluated to assess the quality of the automatically formed clusters, and the performance of IAHCs.

This section illustrated how IAHC is used for software modularization. While, next section gives the literature review of IAHC approaches used for software modularization.

Algorithm 4 - Weighted Average Linkage (WL) Method

**Input:** Similarity Matrix  
**Output:** Updated Similarity Matrix

1: Update similarity matrix by calculating similarity between newly formed cluster and the existing entities or clusters. Let suppose three entities $E_i$, $E_j$ and $E_k$ and the newly formed cluster is $E_iE_j$. Now similarity between $E_k$ and newly formed cluster $E_iE_j$ is calculated as:

$$WL(E_iE_j, E_k) = 0.5 \times (SM(E_i, E_k) + SM(E_j, E_k))$$
2.5 Software Modularization using IAHCs

In order to better understand and to get more insight of the included literatures, this section lists the literature according to chronological order. A summary of all the literatures is presented in a tabular form in Table 2.4.

Davey & Burd (2000) evaluated the IAHCs for software modularization. Authors assessed four IAHCs, i.e., Complete linkage (CL), Single Linkage (SL), Weighted Average Linkage (WL), and Unweighted Average Linkage (UWL), and four similarity measures, i.e., Jaccard (JC), Sorensen Dice (SD), Canberra and Correlation coefficient. Procedures were selected as entities and three features were used: global, user defined types, calls, and combination of all the three. Experiments on three proprietary systems revealed that the CL with JC and SD measures create high quality results in terms of precision and recall with large number of small size clusters. All features produce better results as compared to individual category of features.

To find the stability of the clustering algorithms, Tzerpos & Holt (2000b) compared IAHC-CL and IAHC-SL, using MoJo as an assessment measure. Experimental results with TOBEY and Linux software systems concluded that SL with cut point 90 (SL90) performed better, with 100% stability results, then SL with cut
point 75 (SL75) performed better while CL with cut point 90 (CL90) performed worst. Increasing the cut point height resulted in decreasing the stability, while in case of SL increasing cut point resulted in increasing stability.

Clustering the software entities in a module may increase or decrease the collective information of the module. Based on the given assumption, Andritsos & Tzerpos (2003) proposed a new measure, called Information Loss (IL). IL tries to minimize the loss of information between any two entities while making a new cluster. Agglomerative Information Bottleneck (AIB) (Slonim & Tishby, 2000) has been improved to reduce the time complexity and scalable Information Bottleneck (LIMBO) algorithm was introduced for clustering. To evaluate the applicability of LIMBO using IL, the algorithm was applied to two large size test systems i.e. TOBEY and Linux. LIMBO was compared to ACDC (Tzerpos & Holt, 2000a), Bunch (Mancoridis et al., 1999) and IAHCs algorithms (i.e., CL, SL, WL, and UWL). LIMBO performed better using MoJo as assessment criterion. However, LIMBO based on the non-binary features.

Anquetil & Lethbridge (2003) presented a comparative analysis of clustering algorithms for software modularization. Authors analyzed the selection of features (formal and nonformal) for entity description, measures (Taxonomic, Camberra, JC, Simple Matching, SD, and Correlation) to calculate association between entities, and IAHC algorithms (i.e., CL, SL, WL, and UWL). Different evaluation criteria were used, such as, Precision/Recall for comparison with reference decomposition, Cohesion/Coupling using a similarity measure for design, size of the cluster, Information measure for entropy, and nearest neighbor criterion to find the similarity between two feature types. From the experiments conducted on three open source and one proprietary systems, authors concluded that: 1) on average nonformal features have more information than formal features, 2) 'Comment', a nonformal feature has least redundancy with other feature kinds while 'Identifier' has highest, 3) nonformal features can be used to cluster the systems, 4) using sibling/indirect features for remodularization because these have more capability of feature kinds, 5) similarity metrics that count absence of a feature as a sign of similarity perform worse, and 6)
CL performs better in terms of cohesion while SL performs poorly.

Combined algorithm, a new software clustering algorithm was proposed in (Saeed et al., 2003). This algorithm updates the binary feature vector of a newly formed cluster by taking OR between the two feature vectors of entities making that new cluster. JC similarity measure was used to calculate the similarity between two entities. Authors conducted experiments using Xfig test system and concluded that the cut point of precision and recall for combined algorithm was high as compared to the CL algorithm. The number of clusters produced by the JC was also compared with the SD, Correlation, and Simple measures using CL algorithm. JC and SD produced same results while correlation was very similar to JC. Authors also provided the justification for JC and SD measures, that they are most intuitive for software clustering.

Lung et al. (2004) demonstrated the use of classical clustering techniques and shared the experiences while applying the techniques to software clustering, software restructuring, and maximizing cohesion, and minimizing coupling. Authors added some concluding remarks, including: reverse engineering tools may not create an accurate and complete design and utilities are difficult to deal with, some clusters may contain a very small number of entities or utility entities, cluster analysis may create unexpected results, and finally that techniques are sensitive to the input data therefore more information should be extracted to reach consistent results.

Combined algorithm (Saeed et al., 2003) does not account for access information of a feature by an entity in a cluster. Maqbool & Babri (2004) overcome this limitation by introducing a new algorithm, called Weighted Combined Algorithm (WCA), which keeps information of each entity in a cluster that uses a certain feature. To calculate association between two entities, authors used Unbiased Ellenberg and Gleason similarity measures. The experiments were performed on Xfig test system. Number of clusters and Precision/recall were the evaluation criteria. CL using JC measure, Combined algorithm using JC measure and WCA using Unbiased Ellenberg measure created similar results in terms of number of clusters. The precision/recall crossover heights were found better for Weighted Combined algorithm.

Andritsos & Tzerpos (2005) presented the scalable information bottleneck
(LIMBO) algorithm based on Information Loss (IL) measure. IL measure finds the entropy of two entities while making a new cluster. The algorithm was tested on 3 large sized test systems. LIMBO was compared with ACDC, Bunch (Nearest hill-climbing (NAHC), and steepest-ascend hill-climbing (SAHC)) and IAHC algorithms using MoJo distance metric. LIMBO performed well, if not better than other algorithms. LIMBO has also the added benefit to incorporate the non-structural features of the test systems. Weights can also be assigned using different methods like Mutual Information (MI), Linear Dynamical Systems (LDS), TF.IDF, PageRank and Dynamic Usage. TF.IDF, MI, and Inverse PageRank weighting schemes performed better than others.

Xia & Tzerpos (2005) explored dynamic dependencies between entities as relationships for software clustering. Authors extracted static and dynamic relationships of Mozilla test system and clustered the entities using ACDC, Bunch and IAHC algorithms. Experimental results using MoJoFM revealed that dynamic relationships perform worse than static. However, dynamic relationships can provide a good understanding of the system.

Wu et al. (2005) compared the clustering algorithms ACDC, Bunch, CL75, CL90, SL75 and SL90. Stability, authoritativeness and extremity of cluster were the evaluation criteria. According to the stability criterion SL75, SL90 and ACDC were ranked as high quality, CL75 and CL90 ranked medium and Bunch was ranked low. On the authoritativeness criterion, all algorithms were ranked low, however, CL performed better and Bunch performed worse. On the basis of extremity (non extreme clusters) Bunch was ranked highest, CL90 ranked medium, and CL75, ACDC, SL75 and SL90 ranked as low.

Maqbool & Babri (2007b) presented a comprehensive overview of software architecture recovery using hierarchical clustering. Authors gave a detailed analysis of similarity and distance measures, and identified families of measures. They have shown that members of a family generate same results. Similarity measures are better for software clustering while distance measures may be used for utilities detection. They compared clustering algorithms and showed that arbitrary decisions taken by an algorithm can influence the results. They also concluded that large size of clusters
results in clustering stability. They concluded that LIMBO using information loss measure and WCA using unbiased Ellenberg measure performed better in terms of precision and recall and suggested to use LIMBO if utilities are placed in a single cluster.

Patel et al. (2009) proposed a novel cascaded two phase software clustering technique that integrates static and dynamic analysis of the source code. First phase constructs the core skeleton decomposition using dynamic analysis of the system. In this phase IAHC-CL with JC similarity measure is used to cluster the entities. In the second phase, the remaining entities are clustered using their static dependencies with existing clusters, with the help of the orphan adoption algorithm in (Tzerpos & Holt, 2000a). A case study using Weka as a test system was conducted which showed the usefulness of the proposed technique in terms of MoJoFM.

Muhammad et al. (2010) evaluated eighteen different types of relationships (features) between classes for object oriented software systems. They divided these relationships into direct and indirect types. For example, inheritance between two classes is a direct relationship, but any two child classes extending a parent class represents an indirect relationship between child classes. For clustering the well-known IAHC algorithms were used, that is, CL, WL, and UWL. The objective function was taken to be the count of common relationships between classes. Relationships were evaluated using MoJoFM, on three different industrial software systems. From experimental results, authors concluded that indirect relationships provided better results as compared to direct and the combination of direct and indirect relationships.

Naseem et al. (2010) identified some feature vector cases where the JC similarity measure produced unexpected results. Authors highlighted two feature vector cases to show the deficiency in JC measure and proposed a new similarity measure to solve the problem, called Jaccard-NM (JNM). The new measure was evaluated on three industrial software systems using MoJoFM as assessment criterion. Experimental results revealed the better performance of the JNM over JC measure.

Scanniello et al. (2010b) adopted a variant of K-Means (KM) algorithm for software clustering using Latent Semantic Indexing (LSI). This variant minimizes
REFERENCES


Use of Lexical Information for Software System Clustering. In *European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE. pp. 35–44.


Applications, 39(12), 11303–11311.


