LOCAL SEARCH FOR OPTIMIZING SOFTWARE DESIGN
MIGRATION FROM STRUCTURED PROGRAMMING TO
OBJECT ORIENTED PARADIGM

MD SAEED SIDDIK
BSSE 0218

A Thesis
Submitted to the Bachelor of Science in Software Engineering Program Office
of the Institute of Information Technology, University of Dhaka
in Partial Fulfillment of the
Requirements for the Degree

BACHELOR OF SCIENCE IN SOFTWARE ENGINEERING

Institute of Information Technology
University of Dhaka
DHAKA, BANGLADESH

© MD SAEED SIDDIK, 2013
LOCAL SEARCH FOR OPTIMIZING SOFTWARE DESIGN MIGRATION
FROM STRUCTURED PROGRAMMING TO OBJECT ORIENTED
PARADIGM

MD SAEED SIDDIK

Approved:

Signature

Date

Supervisor: Shah Mostafa Khaled

Committee Member: Dr. Kazi Muheymin-Us-Sakib

Committee Member: Dr. Shariful Islam

Committee Member: Dr. Mohammad Shoyaib
To my beloved father Md. Abu Bakar Siddik who is the best source of inspiration for me
Abstract

Several industries are using legacy softwares, developed with Structured Programming (SP) approach, that should be migrated to Object Oriented Paradigm (OOP) for ensuring better software quality parameters like modularity, manageability and extendability. Automating SP to OOP migration is pivotal as it could reduce time that take in the manual process. Given this potential benefit, the issue is yet to be addressed by researchers. This thesis addresses the scenario by modeling this problem as a graph clustering problem where SP functions and function calls are vertices and edges respectively. The challenge evolving the problem is to find optimized clusters from graphs. To aid this problem, certain heuristic algorithms based on Monte Carlo, Greedy approaches and Local Search are being developed. The proposed algorithms have been tested against a collection of real and synthetic data. The numerical results show that Greedy algorithms are faster than any other proposed heuristic algorithms and Local Search algorithm produced best results among all other proposed approaches.
Acknowledgments

First and above all, I praise Allah, the Almighty and the Guardian, for providing me this opportunity and granting me the capability to proceed successfully. I would like to express my deep and sincere gratitude to my thesis supervisor Mr. Shah Mostafa Khaled, Assistant Professor, IIT University of Dhaka, for providing invaluable guidance throughout my thesis. His dynamism, vision, sincerity and motivation have deeply inspired me. He not only contributed as my technical supervisor but also helped as my psychological mentor. Many thanks to the members of thesis committee Dr. Kazi Muheymin-Us-Sakib and Dr. Mohammad Shoyaib for helping me by providing feedbacks.

Thanks to Mr. Md. Selim, member of IIT DU Optimization Research Group, for his total cooperation. Many thanks to Mr. Alim Ul Gias for his mentoring and technical support. My sincere gratitude to Dr. Shahadat Hossain, Associate Professor, Department Math & Computer Science, University of Lethbridge, AB, Canada for presenting this problem to us. Thanks to Mr. Tahsin Zulkarnine, Web Application Developer, Commerx Corporation, Calgary, AB, Canada for providing experimental data.
### Contents

Approval ........................................ ii  
Dedication ....................................... iii  
Abstract .......................................... iv  
Acknowledgments ................................... v  
Table of Contents ................................ vi  
List of Tables ...................................... vii  
List of Figures .................................... viii

1 Introduction ..................................... 1  
   1.1 The Problem .................................. 1  
   1.2 Research Questions ........................... 3  
   1.3 Organization of this Thesis ................. 4

2 Background ....................................... 5  
   2.1 Structured Programming ....................... 5  
   2.2 Object Oriented Programming ................... 5  
   2.3 Legacy Software and Legacy Code ............. 6  
   2.4 Basic Graph Terminology ...................... 6  
   2.5 Call Graph .................................... 8  
   2.6 Design Structure Matrix ....................... 10  
   2.7 Graph Clustering .............................. 11  
   2.8 Cluster Validity Index ....................... 11  
      2.8.1 Clustering Coefficient .................... 12  
      2.8.2 Characteristics Path Length .............. 12  
   2.9 Summary ...................................... 13

3 Literature Review ................................ 14  
   3.1 Design Migration .............................. 14  
   3.2 Graph Clustering .............................. 15  
   3.3 DSM Partitioning ............................... 16  
   3.4 Heuristics Algorithm ........................... 17  
   3.5 Summary ...................................... 18

4 Heuristic Algorithms for Software Design Migration .... 19  
   4.1 Mathematical Formulation of the Problem .......... 19  
   4.2 Proposed Heuristic Algorithms ................. 21  
      4.2.1 Monte Carlo Algorithm ..................... 22  
      4.2.2 Greedy Algorithm .......................... 25  
      4.2.3 Local Search Algorithm ..................... 26
5 Experimental Results

5.1 Environmental Setup ........................................... 28
5.2 Dataset Description ............................................ 28
5.3 Runtime Report .................................................. 29
5.4 Result of Monte Carlo Algorithm .............................. 29
  5.4.1 Dataset BFT .................................................. 30
  5.4.2 Dataset RBIo .................................................. 31
  5.4.3 Dataset Synthetic7 ......................................... 32
  5.4.4 Dataset Synthetic71 ........................................ 33
  5.4.5 Dataset Synthetic100 ...................................... 34
  5.4.6 Dataset Synthetic166 ...................................... 35
5.5 Result of Greedy Algorithm .................................... 36
5.6 Result of Local Search Algorithm ............................ 37
  5.6.1 Dataset BFT .................................................. 37
  5.6.2 Dataset RBIo .................................................. 38
  5.6.3 Dataset Synthetic71 ........................................ 38
  5.6.4 Dataset Synthetic100 ...................................... 39
  5.6.5 Dataset Synthetic166 ...................................... 40
5.7 Comparative Result ............................................. 41
5.8 Summary .......................................................... 43

6 Conclusion .......................................................... 44

Bibliography .......................................................... 46

A Comparison Chart on Experimental Results .................. 50

B Monte Carlo Algorithms Result of 100 Execution ........... 57
### List of Tables

1.1 Example of a SP legacy code, its Call Graph and Optimal Clustering 3
2.1 Example of an SP code ................................................. 9
2.2 Design Structured Matrix of the Call Graph, presented in Figure 2.2 10
5.1 Dataset Description including number of vertex and edges ........ 28
5.2 Running Time (in Microseconds) .................................... 29
5.3 Clustering Coefficient ($\Psi$) of Greedy-1 and Greedy-2 ............ 36
5.4 Characteristics Path Length ($\chi$) of Greedy-1 and Greedy-2 .... 36
5.5 Kal-Index ($\kappa$) of Greedy-1 and Greedy-2 ..................... 36
5.6 MC-1 - MC-3 and Greedy-1-Greedy-2 and Local Search Algorithms on BFT ................................................................. 41
5.7 MC-1 - MC-3, Greedy-1-Greedy-2 and Local Search Algorithms RBIo 42
5.8 MC-1 - MC-3, Greedy-1-Greedy-2 and Local Search Algorithms on Synthetic7 ................................................................. 42
5.9 MC-1 - MC-3, Greedy-1-Greedy-2 and Local Search Algorithms on Synthetic71 ................................................................. 42
5.10 MC-1 - MC-3, Greedy-1-Greedy-2 and Local Search Algorithms on Synthetic100 ........................................................... 42
5.11 MC-1 - MC-3, Greedy-1, Greedy-2 and Local Search Algorithms on Synthetic166 .............................................................. 43
# List of Figures

2.1 Example of Undirected and Directed Graph .................................................. 7
2.2 Call Graph corresponding to the code in Table 2.1 ........................................... 9
2.3 Example Graph Clustering ............................................................................. 11

5.1 Monte Carlo Algorithms on \textit{BFT} ................................................................. 30
5.2 Monte Carlo Algorithms on \textit{RBlo} ................................................................. 31
5.3 Monte Carlo Algorithms on \textit{Synthetic7} ........................................................ 32
5.4 Monte Carlo Algorithms on \textit{Synthetic71} ...................................................... 33
5.5 Monte Carlo Algorithms on \textit{Synthetic100} .................................................... 34
5.6 Monte Carlo Algorithms on \textit{Synthetic166} .................................................... 35
5.7 Improvement of Kal(\(\kappa\))-index using Local Search for dataset \textit{BFT} 37
5.8 Improvement of Kal(\(\kappa\))-index using Local Search for \textit{RBlo} ............... 38
5.9 Improvement of Kal(\(\kappa\))-index using Local Search for \textit{Synthetic71} ....... 39
5.10 Improvement of Kal(\(\kappa\))-index using Local Search for \textit{Synthetic100} .... 40
5.11 Improvement of Kal(\(\kappa\))-index using Local Search for \textit{Synthetic166} .... 40

A.1 Huristics Algorithms Results Comparison on \textit{BFT} ....................................... 51
A.2 Huristics Algorithms Results Comparison on \textit{RBlo} ......................................... 52
A.3 Huristics Algorithms Results Comparison on \textit{Synthetic7} ............................ 53
A.4 Huristics Algorithms Results Comparison on \textit{Synthetic71} ......................... 54
A.5 Huristics Algorithms Results Comparison on \textit{Synthetic100} ....................... 55
A.6 Huristics Algorithms Results Comparison on \textit{Synthetic166} ....................... 56

B.1 Clustering Coefficient(\(\Psi\)) on \textit{BFT} ......................................................... 58
B.2 Characteristics Path Length(\(\chi\)) of \textit{BFT} ................................................. 59
B.3 KAL index (\(\kappa\)) of \textit{BFT} ................................................................. 60
B.4 Clustering Coefficient(\(\Psi\)) on \textit{RBlo} ......................................................... 61
B.5 Characteristics Path Length(\(\chi\)) of \textit{RBlo} .............................................. 62
B.6 KAL index (\(\kappa\)) of \textit{RBlo} ................................................................. 63
B.7 Clustering Coefficient(\(\Psi\)) on \textit{Synthetic71} ........................................... 64
B.8 Characteristics Path Length(\(\chi\)) of \textit{Synthetic71} .................................... 65
B.9 KAL index (\(\kappa\)) of \textit{Synthetic71} ....................................................... 66
B.10 Clustering Coefficient(\(\Psi\)) on \textit{Synthetic100} ........................................ 67
B.11 Characteristics Path Length(\(\chi\)) of \textit{Synthetic100} .................................. 68
B.12 KAL index (\(\kappa\)) of \textit{Synthetic100} ....................................................... 69
B.13 Clustering Coefficient(\(\Psi\)) on \textit{Synthetic166} ........................................ 70
B.14 Characteristics Path Length(\(\chi\)) of \textit{Synthetic166} .................................. 71
B.15 KAL index (\(\kappa\)) of \textit{Synthetic166} ....................................................... 72
Chapter 1

Introduction

A good number of software are still being used in industries which were developed decades earlier using Structured Programming (SP) approach. These software often go out of support and termed as Legacy Software. The source code of such software is commonly referred as Legacy Code which becomes unmanageable when it grows too large [1]. Large and complex programs are easier to maintain if developed with Object Oriented Paradigm (OOP) that offers better re-usability, modularity, manageability and extendability by means of encapsulation, inheritance and polymorphism. This research focuses to migrate the existing SP legacy code to OOP architecture.

1.1 The Problem

In case of bug solving or feature enhancement and integration, legacy codes demand much more time than OOP ones. Such difficulties have their consequential business impacts and thus many industries, dependent on legacy softwares, are facing the issue of business sustainability. Possible way outs could be re-designing the whole product from the scratch or manual SP to OOP design migration which could be error-prone and time consuming. This research intends to propose an approach for automatic SP to OOP design migration. The problem was presented as one of the open problems at 13th International Dependency and Structure Modeling Conference (DSM 2011), Cambridge, Massachusetts, USA.

To analyze the problem we formulate the scenario in terms of a graph where each function written in SP code is represented as a vertex, and a function call is represented as a directed edge. This gives us a connected directed graph referred as Call Graph [2, 3]. The adjacency matrix representation of the call graph produces
a 0-1 matrix. The matrix is known as Design Structure Matrix or Dependency Structure Matrix (DSM) [4] which is widely used in industrial engineering for a good variety of applications.

The hypothesis that the methods of a same class will call each other more frequently than the number of calls between methods of different classes has been used on the call graph. This hypothesis is based on encapsulation [5] that says attributes and methods of same class or interface are more interrelated than attributes and methods of different classes or interfaces [6]. The call graph has an underlying undirected graph which is used to search for vertex clustering having objectives to maximize number of intra-cluster edges and minimize number of inter-cluster edges.

An optimal clustering of the call graph produces number of non-overlapping vertex clusters that represent groups of closely related functions in the SP code. This is used as the clue for migrating the underlying SP design to OOP, where each cluster in the call graph represents a class. The migration is simple as calls between methods of the same class do not change in the new design. However, calls between methods of two different clusters has to be re-written incorporating the issue of making call through proper interfaces between classes.

A mathematical definition of the optimal cluster finding problem has been formulated in terms of an Integer Program (IP)[7]. It has been realized that the solution to the IP is computationally hard [8], and therefore different heuristics are applied to get a near optimal solution. A number of randomized and greedy heuristic algorithms has been proposed to assist optimal clustering. The objective function of IP along with conventional Clustering Co-efficient (CC) [9] and Characteristics Path Length(CPL) [10] are used to assess quality of clustering.

Table 1.1 presents a sample SP legacy code, its corresponding call graph, underlying undirected graph of the call graph and an optimal clustering in of the undirected graph.
1.2 Research Questions

As described in the previous section, the nature of Design Migration problem raises the following questions:

- Are the existing technologies sufficient to migrate a Structured[1] Legacy Code to Object Oriented Paradigm?

- If the existing technologies are not supportive enough, than what approach is needed to migrate those programs?

- How to find optimal classes/interfaces from new migrating architecture?
The main aim of this research is to answer the questions mentioned above and thus providing a solution for migrate a Structured Program to Object Oriented Paradigm.

1.3 Organization of this Thesis

Rest of the thesis is organized as follows: Chapter 2 reviews the background of the problem and Chapter 3 focused on the relevant work available in existing literature. A mathematical formulation of the problem is presented in Chapter 4. Chapter 4 also presents three variations of Monte Carlo based heuristics, two variations of Greedy algorithms and a Local Search based heuristic. Numerical results and analysis has been presented in Chapter 5. Chapter 6 concludes the thesis with future research direction.
Chapter 2

Background

OOP seems to be more advanced than SP languages in terms of technology, but it is also more complex than SP [11]. The SP codes have been developed based on computational problem, however OOP evolved due to need for better modularity and maintainability. This Chapter presents the definition and other information necessary to understand this thesis.

2.1 Structured Programming

Structured programming is a programming paradigm which used to improve the quality, lucidity, and development time of a computer program [12, 13]. This programming concept is focused mainly on computational time improving and memory management issues. It is the foundation of Modular Programming and Object Oriented Programming, as it is assumed the individual methods are structured [14]. It is possible to implement structured programming concept in any programming language, though it is preferable to use in procedural programming [15] languages for example C, C++, Fortan and Pascal. These programs become unmanageable when they grow too large because of their low-modularity and low-maintainability features.

2.2 Object Oriented Programming

OOP is a design philosophy that stands for Object Oriented Programming [16, 17]. OOP uses a different set of programming languages than old structured and procedural programming languages (C, Pascal, etc.) [14]. Everything in OOP is grouped as self sustainable objects. Those Objects are considered as interfaces or
classes [16]. In common speech one refers to a file as a class, while the file itself is the object.

In order to clearly understand the object orientation, we can take our hand as an example. The hand is a class. Our body has two objects of type hand, named left hand and right hand. Their main functions are controlled/managed by a set of physical signals sent through the shoulders (through an interface). So the shoulder is an interface which the body uses to interact with the hands. The hand is a well architectural class. The hand is being re-used to create the left hand and the right hand by slightly changing the properties of it.

2.3 Legacy Software and Legacy Code

A good number of software are still being used in industries which were developed decades earlier using Structured Programming (SP) approach. These software often go out of support and termed as Legacy Software [18]. Legacy software are written years ago using outdated technique including structured programming and low modularity procedure [1, 19]. Those softwares continue to do useful work. Many of industries are still using those types of software as in house maintenance and production support. Migrating and updating this legacy software needs a lot of technical and non-technical challenges [20]. The source code of such software is commonly referred as Legacy Code. Legacy Code become unmanageable when it grows too large and quite impossible to integrate new module or manage for extra features [21].

2.4 Basic Graph Terminology

A graph $G = (V, E)$ is a representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by
mathematical abstractions called \textit{vertices} \(v \in V\), and the links that connect some pairs of vertices are called \textit{edges} \(e \in E\) [22]. The vertices connecting to an edge are called the ends, or endpoints of the edge. A vertex may exist in a graph and not belong to an edge. Graphs are used to represent networks of communication, data organization, computational devices, the flow of computation, etc. In this thesis we modeled the problem as graph clustering problem where every functions are presented by vertices \(V\) and every calls are represented by edges \(E = \{V_i, V_j\}\).

![Diagram](image)

\textbf{Figure 2.1: Example of Undirected and Directed Graph}

Graphs can be directed or undirected. In an undirected graph, the vertices \((u, v) \in e\) are unordered. That means the pair \((u, v)\) and \((v, u)\) represents the same edge. When an edge \(e = (u, v)\) in directed graph, \(u\) is the tail and \(v\) is the head of the edge. So, the pairs \((u, v)\) and \((v, u)\) represent two different edges in the graph. If the tail and head are the same vertices for an edge, then its called a loop. Directed graphs are also called digraphs.

In Fig. 2.4, vertices are shown in circles and edges are the lines connecting the vertices. A graph \(G = (V, E)\) in 2.4.a, where \(V = 1, 2, 3, 4, 5, 6\) and \(E = (1, 3),\)
(1, 4), (1, 5), (2, 4), (2, 6), (3, 5), (4, 6). In 2.4.b, a graph \( G = (V, E) \), \( G \) has the same set of vertices \( V \) as 2.4.a but \( E = (1, 3), (1, 4), (2, 6), (3, 5), (3, 5), (4, 2), (5, 1), (5, 3), (6, 2), (6, 4) \).

### 2.5 Call Graph

A call graph is a directed graph that represents calling relationships between modules or functions in a computer program[23, 24, 2]. To analyze the problem we formulate the research in terms of a graph where each function written in SP code is represented as a vertex, and a function call is represented as a directed edge. This gives us a connected directed graph referred as Call Graph [2, 3].

Specifically, each node represents a function and each edge \((f, g)\) indicates that function \( f \) calls function \( g \). Thus, a cycle in the graph indicates recursive function calls. Call graph is a basic tool that can be used for analysis of program architecture and human understanding of programs. Call graphs can be dynamic or static[25]. A dynamic call graph is a record of an execution of the program, e.g., as output by a profiler. Thus, a dynamic call graph can be exact, but only describes one run of the program. A static call graph is a call graph intended to represent every possible run of the program. The exact static call graph is un-decidable problem, so static call graph algorithms are generally over approximations. In our research we collect dynamic call graph for real life data set and static call graph for synthetic dataset.
Table 2.1 provides an example of Structure Programming code where 5 functions have been executed and 9 calls between those functions. The call graph of Table 2.1 is given below:

Table 2.1: Example of an SP code

<table>
<thead>
<tr>
<th>functionA{</th>
<th>functionB{</th>
<th>functionD{</th>
</tr>
</thead>
<tbody>
<tr>
<td>......</td>
<td></td>
<td></td>
</tr>
<tr>
<td>functionB();</td>
<td></td>
<td></td>
</tr>
<tr>
<td>functionC();</td>
<td></td>
<td></td>
</tr>
<tr>
<td>functionB();</td>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
<td>functionA();</td>
</tr>
<tr>
<td></td>
<td>functionC{</td>
<td></td>
</tr>
<tr>
<td></td>
<td>......</td>
<td></td>
</tr>
<tr>
<td></td>
<td>functionD();</td>
<td></td>
</tr>
<tr>
<td></td>
<td>functionD();</td>
<td></td>
</tr>
<tr>
<td></td>
<td>functionE();</td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>functionE{</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>..</td>
<td>functionD();</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

Figure 2.2: Call Graph corresponding to the code in Table 2.1
2.6 Design Structure Matrix

The Design Structure Matrix (DSM) (also known as dependency structure matrix, dependency structure method, dependency source matrix, problem solving matrix (PSM)) is a simple, compact and visual representation of a system or project in the form of a \((n \times n)\) matrix [26]. The cells along the diagonal represent the system elements, which are often labeled in the rows to the left of the matrix and/or in the columns above the matrix. The off-diagonal cells are used to indicate relationships between the elements[27]. DSM can represent a large number of system elements and their relationships in a compact way that highlights important patterns. Two main categories of DSMs have been proposed: static and time-based. Static DSMs represent systems where all of the elements exist simultaneously, such as components of a machine or groups in an organization. Static DSMs are usually analyzed with clustering algorithms[4].

The Design Structure Matrix or dependency structure matrix (DSM) of the Call Graph Figure 2.2 is attached below:

Table 2.2: Design Structured Matrix of the Call Graph, presented in Figure 2.2

<table>
<thead>
<tr>
<th>function</th>
<th>fA</th>
<th>fB</th>
<th>fC</th>
<th>fD</th>
<th>fE</th>
</tr>
</thead>
<tbody>
<tr>
<td>fA</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fB</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>fC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>fD</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
2.7 Graph Clustering

The process of identifying the underlying structure of a program in terms of grouping the data elements is called clustering, also called data classification [28]. The resulting groups are called clusters. The grouping is usually based on some similarity measure defined for the data elements. Graphs are structures formed by a set of vertices (also called nodes) and a set of edges that are connections between pairs of vertices. Graph clustering is the task of grouping the vertices of the graph into clusters taking into consideration the edge structure of the graph in such a way that there should be many edges within each cluster and relatively few between the clusters [29]. In the Figure 2.3, 18 nodes are covered by 3 clusters. Each vertex or nodes are allocated in a single cluster.

![Figure 2.3: Example Graph Clustering](image)

2.8 Cluster Validity Index

Cluster validity indexes is the validation measurement of any cluster. Halkidi et al. [30] claimed that evaluating and assessing the results of a clustering algorithm is the main subject of cluster validity. Bezdek et al. [31] presented a various number of new indexes of cluster validity.
2.8.1 Clustering Coefficient

In graph theory, a clustering coefficient [9] is a measure of degree to which nodes in a graph tend to cluster together. In our research Local clustering coefficient is used to measure Clustering Coefficient (Ψ) index. The local clustering coefficient of a vertex (node) in a graph quantifies how close its neighbors are to being a Complete Graph [9]. A graph \( G = (V,E) \) consists of a set of vertices \( V \) and a set of edges \( E \) between them. An edge \( e_{ij} \) connects vertex \( v_i \) with vertex \( v_j \). The neighborhood \( N_i \) for a vertex \( v_i \) is defined as its immediately connected neighbors as follows: \( N_i = \{ v_j : e_{ij} \in E \cap e_{ij} \in E \} \). Clustering Coefficient \( \Psi \) of an undirected graph is defined by,

\[
\Psi = \frac{1}{N} \sum_{i=1}^{N} \Psi_i \quad \text{where} \quad \Psi_i = \frac{2 \{ e_{ij}, e_{jk} \in E \} \cap N_i \cap N_j}{k_i(k_i-1)}
\] (2.1)

Equation 2.1 denotes the clustering coefficient (CC) of node \( i \), with \( k_i \) being the number of nodes connected to node \( i \), and \( n_i \) being the actual number of edges between those \( k_i \) adjacent nodes.

2.8.2 Characteristics Path Length

In any connected undirected graph, the average distance between pairs of vertices is defined by Characteristics Path Length (CPL)[10]. In an unweighed undirected graph \( G \) with the set vertex \( V \). Let \( d(v_i, v_j) \), where \( \{ v_1, v_2 \} \in V \) denote the shortest distance between nodes \( v_i \) and \( v_j \). Assume that \( d(v_i, v_j) = 0 \) if \( v_1 = v_2 \) or \( v_2 \) cannot be reached from \( v_1 \). Than the CPL \( \chi \) is:

\[
\chi = \frac{1}{N(N-1)} \cdot \sum_{i \neq j} d(v_i, v_j)
\] (2.2)
2.9 Summary

For migrating a design we need to draw the call graph of the legacy code and generate the DSM from the call graph. Then execute the proposed heuristics to find optimal cluster and measure the validity index of those clusters. This chapter describes the background of this work and present the detail explanation of those terminologies which have been used frequently in this thesis.
Chapter 3

Literature Review

Automatic SP to OOP design migration has been rarely addressed as a direct research problem in the existing literature. Although plenty of work has been done with Graph Clustering [29] and DSM [4], existing graph clustering methods mainly focus on the Euclidean distance, but largely ignore vertex connectivity where every distances between two vertices’s are same. On the other hand, the researches involving DSM focuses on searching function calls that can be triggered parallely. This chapter presents the literature review of this problem including design migration, graph clustering, DSM partitioning, heuristics algorithms etc.

3.1 Design Migration

Automatic migration from code to design was introduced in [32]. The work converts a COBOL program to a Object Oriented design document. Different types of objects like user interface objects, information objects, file and view objects etc in existing COBOL systems has been identified in this research. The final steps was to capture and document the sequence of operations which were executed in existing COBOL system. This work has been rendered for migrating COBOL program but not for migrating structured C program.

Maqbool et al. [33] reviewed hierarchical clustering research in the context of software architecture recovery and modularization. A detailed analysis of the behavior of different similarity and distance measures for software clustering has been presented in that paper. Moreover, their work analyzed the clustering process of multiple clustering algorithms using multiple criteria and showed how arbitrary decisions taken by these algorithms affect the quality of the clusters.

Heroux et al. [34] presented the architectural comparison of commercial soft-
ware and scientific research software. This paper reported that commercial softwares are written for the purpose of generating revenue where the underlying algorithms and methodologies are mature but the scientific research softwares in computational science and engineering disciplines are developed for new algorithms and computational capabilities which are less modular and largely unaware of standard industry concepts and practices. Finally they proposed a design approach to convert those scientific research softwares in modular.

In 2011 Dineshkumar et al. [35] presented an empirical approach to migrate from Structured Programming Code to Object Oriented Design. Their work introduced a new technique for code to design migration which creates agglomerative cluster using Jaccard distance matrices. They initialized the relationship between variables and functions of the structured program using Jaccard distance measure that leads to grouping or clustering. They represented the result in Unified Modeling Language (UML) and collect some industrial survey for their proposal accuracy.

3.2 Graph Clustering

Zhou et al. [36] proposed a graph clustering algorithm named SA-Cluster based on similar attribute using unified distance measure. This method partitions a large graph associated with attributes into $k$-clusters so that each cluster contains a densely connected sub-graph with homogeneous attribute values. An effective method is proposed to automatically learn the degree of contributions of structural similarity and attribute similarity has been proposed in this paper which demonstrate the effectiveness of SA-Cluster through comparison with the state-of-the-art graph clustering and summarization methods.

Aggarwal et al. [37] proposed an algorithm to find appropriate sets of clusters and dimensions using medoids technique based on Euclidean coordinate points
and preceded by feature selection.

Bezdek et al. reviewed two clustering algorithms (hard c-means [38] and single linkage) and three indexes of crisp cluster validity (Hubert's statistics, the Davies-Bouldin index, and Dunns index) [31]. Their work illustrates two deficiencies of Dunns index [39] that make it overly sensitive to noisy clusters. They proposed several generalizations of those deficiencies which are not as brittle to outliers in the clusters. Definitions regarding cluster in a graph and measures of cluster quality were reviewed in [29]. This work also presented global algorithms for clustering the entire vertex set of an input graph and discussed the task of identifying a cluster for a specific seed vertex by local computation [40].

3.3 DSM Partitioning

Hossain et al. presented an analytical report on design structure of open source scientific computing software [27]. They used a number of architectural complexity metrics and DSM technique to analyze the design structure. Their analysis involved Automatic Differentiation (AD), Linear Programming (LP) and Mixed Integer Programming (MIP). They have used DSM to present functions that are explicitly implemented in the software under consideration (denoted as user function) and functions that are part of software libraries. Those DSM qualities are measured by characteristic path length, clustering co-efficient, nodal degree, strongly connected components, propagation cost, etc. [10].

Sullivan et al. [41, 42] presented a potential new theory to account the influence of modularity on the evolution of software design. That theory uses design structure matrices to model designs and real options techniques to value those model.
3.4 Heuristics Algorithm

Heuristics are the most exciting development in approximate optimization techniques of the last two decades. A heuristic is formally defined as an iterative generation process which guides a subordinate heuristic for exploring and exploiting the solution space. Different types of learning strategies are used to structure information in order to find efficiently near-optimal solutions[43, 44]. Osman et al. [45] presented the comparison of different heuristic techniques which were applied on various problem. Several classes of simulation optimization problems and solution methodologies have been analyzed there.

Monte Carlo algorithm [46] is a randomized algorithm that may fail or return an incorrect answer. The running time of a Monte Carlo algorithm often does not depend on the random choices made. Generating a good sequence of samples is often a nontrivial task and is a major focus of the Monte Carlo method. Monte Carlo methods may vary to find solution, but tend to follow a particular pattern. Although the best solution of Monte Carlo may not be the optimal solution.

A Greedy Algorithm is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum[47]. Angel-Bello et al. [48] claimed that in many problems, a greedy strategy does not in general produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a global optimal solution in a reasonable time. That research proposed a new Greedy Randomize procedure to find the optimal solution in large search space. Greedy Random Adaptive Search Procedure (GRASP) has been presented by Resende et al.[49] which was one of the modified version of generic greedy search. GRASP was a multi-start heuristic for combinatorial problem in which each iteration consists of construction and search phases.

A Local Search Algorithm is one of the proposed heuristic which starts from
an initial solution seed and iterates exploring the solution spaces, using the moves or updates associated with the neighborhood definition. This method is widely used for solving computationally hard optimization problems[50]. Van-Laarhoven et al. [51] used local search for the resolution bias most objective function matters in graph clustering, Korupolu et al. [52] used local search heuristic on facility location problems, etc.

3.5 Summary

In this chapter we have described the background literature review of our problem. Review of existing literature shows that none of the work directly proposed a method that focus on converting SP code to detailed object oriented design. Those approaches also not yet used to solve design migration problem. We have proposed new approaches to migrate SP code to OOP design by means of optimal clustering form a call graph. To measure the quality of the clusters, a new proposed metric will be discussed in next chapter.
Chapter 4

Heuristic Algorithms for Software Design Migration

This chapter presents a mathematical formulation of the Structured to Object Oriented Paradigm design migration problem. The problem is a computationally hard problem therefore we have developed few heuristic algorithms on it. Those algorithms are also provide in this chapter.

4.1 Mathematical Formulation of the Problem

Let $G = (V, E)$ be the underlying undirected graph of a call graph. $V$ and $E$ be the set of vertices and edges respectively, with $n = |V|, m = |E|$. We define variables $x_e$ and $y_e$ corresponding to each edge $e \in E$, $C_v$ corresponding to each vertex $v \in V$. $G$ may have at most $n$ clusters, where each vertex in the graph will be in a distinct cluster. A vertex, therefore, may potentially be in one of $n$ clusters. The variables are defined as follows:

\[
x_e \in \begin{cases} 
1 & \text{if } e \text{ is an intra-cluster edge} \\
0 & \text{otherwise}
\end{cases}
\]

\[
y_e \in \begin{cases} 
1 & \text{if } e \text{ is an inter-cluster edge} \\
0 & \text{otherwise}
\end{cases}
\]
The problem of maximizing intra-cluster edges, minimizing inter-cluster edges, and maximizing the number of clusters can be formulated as follows:

\[
\text{Maximize } \sum_i x_i - \sum_i y_i + \sum_j |C_j| \quad \forall i=1,2,\ldots,m \text{ and } j=1,2,\ldots,n \quad (4.1)
\]

Subject to:

\[
x_i + y_i = 1 \quad \forall i=1,2,\ldots,m \quad (4.2)
\]

\[
\sum_{l=1}^{n} z_{kl} = 1 \quad \forall k=1,2,\ldots,n \quad (4.3)
\]

\[
\sum_{l=1}^{n} z_{kl} \cdot z_{kl} = x_{(a,b)} \quad \forall (a,b) \in E \quad (4.4)
\]

\[
C_l = \bigcup_k z_{kl} \quad \forall k=1,2,\ldots,n \quad (4.5)
\]

The objective in Equation (4.1) maximizes the number of intra-cluster edges.
and clusters in the graph, and minimizes the number of inter-cluster edges. Equation (4.2) ensures that an edge can be exclusively an intra- or inter-cluster edge. Equation (4.3) ensures that each vertex must belong to a cluster. Equation (4.4) ensures that an intra-cluster edge must have its both endpoints belonging to the same cluster. Equation (4.5) defines the variable $C_j$ as a cluster head, if any vertex belongs to its respective cluster.

The proposed Equation (4.1) will be used as Kal Index to numerically measure the quality of a modular object oriented design. This has been used to judge the quality of output that an algorithms have produced.

4.2 Proposed Heuristic Algorithms

This section presents the Monte Carlo, Greedy and Local Search based heuristic algorithms that we have developed. Experimental results on these algorithms have been presented in the next chapter.

Algorithm 1 Algorithm MC-1

Input: Call Graph $G = (V, E)$

Output: Clustering $C$, Kal $(\kappa)$, Clustering Coefficient $(\Psi)$, Characteristics Path Length $(\chi)$

1: Begin
2: Assume we have $n = \sqrt{|V|}$ number of clusters $C_1, C_2, C_3, \ldots, C_n$
3: for all vertex $v \in V$ do
4: Generate a random number $i \in [0, |V|]$ for vertex $v$
5: Assign $v$ to cluster $C_i$
6: end for
7: Calculate $\kappa$, $\Psi$, $\chi$ using clustering scheme $C$ and equations (4.1), (2.1), and (2.2) respectively
8: End
Algorithm 2 Algorithm MC-2

Input: Call Graph $G = (V, E)$

Output: Clustering $\mathcal{C}$, $\text{Kal}(\kappa)$, Clustering Coefficient($\Psi$), Characteristics Path Length ($\chi$)

1: Begin
2: $\vartheta \leftarrow V$
3: while $\vartheta \neq \emptyset$ do
4: Randomly pick vertex $v \in \vartheta$
5: $\omega$ be the set of clusters, $\exists u \in \omega_j (u, v) \in E$, $\forall \omega_j \in \omega$
6: if $\omega = \emptyset$ then
7: Create new cluster $C_i$
8: $C_i \leftarrow C_i \cup \{v\}$
9: else
10: Randomly pick $C_i \in \omega$
11: $C_i \leftarrow C_i \cup \{v\}$
12: end if
13: $\vartheta \leftarrow \vartheta \setminus \{v\}$
14: end while
15: Calculate $\kappa$, $\Psi$, $\chi$ using clustering scheme $\mathcal{C}$ and equations (4.1), (2.1), and (2.2) respectively
16: End

4.2.1 Monte Carlo Algorithm

Algorithm 1 presents the first variation on Monte Carlo Algorithm that assumes a fixed number of clusters ($\mathcal{C}$). In this approach the desired classes are deterministic and the approximate number of classes are $\lfloor \sqrt{|V|} \rfloor$. After initializing the fixed clusters assign each vertex $v \in V$ to those clusters $C_i \in \mathcal{C}$ randomly. This procedure generate a set of clusters which will be clue to classes or interfaces of the solution.

Algorithm 2 presents the second variation on Monte Carlo method where the number of clusters are not predefined that means the approximate classes or interfaces may vary from 1 to $|V|$. This algorithm randomly picks a vertex $v \in V$ and assign to its first neighbor cluster $C_i \in \mathcal{C}$. If more than one neighbor clusters are present than randomly assign to one cluster. If no cluster exists than the vertex is made the head of a newly created cluster.

Algorithm 3 presents the third and final variation on Monte Carlo Algorithm, also generates scalable number of clusters. This algorithm randomly selects a
Algorithm 3 Algorithm MC-3

**Input:** Call Graph $G = (V, E)$

**Output:** Clustering $C$, Kal($\kappa$), Clustering Coefficient($\Psi$), Characteristics Path Length ($\chi$)

1: Begin
2: while All vertex $v \in V$ are not assigned to any cluster do
3: Randomly pick vertex pair $\{v_1, v_2\} \in V$
4: if $v_1 \in C_i$ and $v_2$ unassigned to any cluster then
5: $C_i \leftarrow C_i \cup \{v_2\}$
6: else if $v_1$ unassigned to any cluster and $v_2 \in C_j$ then
7: $C_j \leftarrow C_j \cup \{v_1\}$
8: else if $v_1$ and $v_2$ both unassigned to any clusters then
9: for all $\nu_i \in \{v_1, v_2\}$ do
10: $\omega$ be the set of clusters, $\exists_u \omega_j (u, \nu_i) \in E$, $\forall \omega_j \in \omega$
11: if $\omega = \phi$ then
12: Create new cluster $C_i$
13: $C_i \leftarrow C_i \cup \{\nu_i\}$
14: else
15: Randomly pick $C_i \in \omega$
16: $C_i \leftarrow C_i \cup \{\nu_i\}$
17: end if
18: end for
19: end if
20: end while
21: Calculate $\kappa$, $\Psi$, $\chi$ using clustering scheme $C$ and equations (4.1), (2.1), and (2.2) respectively
22: End

Algorithm 4 Algorithm Greedy-1

**Input:** Call Graph $G = (V, E)$ Characteristics Path Length

**Output:** Clustering $C$, Kal($\kappa$), Clustering Coefficient($\Psi$), Characteristics Path Length ($\chi$)

1: Begin
2: $\nu \leftarrow V$
3: for all vertex $v \in \nu$ in decreasing order of vertex degree do
4: if $(v, u_i) \in E$ and $u_i \in C_j$ : for any $i = 1...|V|, j = 1...|C|$ then
5: $C_j \leftarrow C_j \cup \{v\}$
6: else
7: Create new cluster $C_k$
8: $C_i \leftarrow C_i \cup \{v\}$
9: end if
10: $\nu \leftarrow \nu \setminus \{v\}$
11: end for
12: Calculate $\kappa$, $\Psi$, $\chi$ using clustering scheme $C$ and equations (4.1), (2.1), and (2.2) respectively
13: End
Algorithm 5 Algorithm Greedy-2

Input: Call Graph $G = (V, E)$

Output: Clustering $C$, $\text{Kal}(\kappa)$, Clustering Coefficient($\Psi$), Characteristics Path Length ($\chi$)

1: Begin
2: $\nu \leftarrow V$
3: for all vertex $v \in \nu$ in decreasing order of vertex degree do
4: if $v \notin C_i$, $i = 1...|C|$ then
5: Set $\nu_1 := \phi$
6: $\nu_1 \leftarrow \nu_1 \cup \{v\}$
7: for all $(v, u_j) \in E$, $j = 1...|V|$ do
8: if $u_j \notin C_k$, $k = 1...|C|$ then
9: $\nu_1 \leftarrow \nu_1 \cup \{u_j\}$
10: end if
11: end for
12: Create new cluster $C'$
13: for all $(v_j) \in \nu_1$ do
14: $C' \leftarrow C' \cup \{v_j\}$
15: end for
16: $\nu \leftarrow \nu \setminus \nu_1$
17: end if
18: end for
19: Calculate $\kappa$, $\Psi$, $\chi$ using clustering scheme $C$ and equations (4.1), (2.1), and (2.2) respectively
20: End

Algorithm 6 Local Search

Input: An initial solution $C_{init}$

Output: Local Optimal Solution $C_{LocOpt}$ better or equal to $C_{init}$

1: Begin
2: $C \leftarrow C_{init}$
3: repeat
4: $C_{LocOpt} \leftarrow \text{UpdateSolution}(C, 1)$
5: if $\text{kal}(C_{LocOpt}) \geq \text{kal}(C)$ then
6: $C \leftarrow C_{LocOpt}$
7: else
8: no change
9: end if
10: until no update made over $x$ iteration of loop
11: $C_{LocOpt} \leftarrow C$
12: return $C_{LocOpt}$
13: End
Algorithm 7 UpdateSolution

Input: Solution $C$ and iteration-number
Output: Best feasible solution $C$ in neighborhood

1: Begin
2: Select a temp solution $C' \leftarrow C$
3: for all $C_i \in C'$, $C' \subseteq C$ and $i = 1, 2, ...|C|)$ do
4:   for all $c_{ij} \in C_i$, $j = 1, 2, ...|C_i|)$ do
5:     $C_i \leftarrow C_i \setminus c_{ij}$
6:     for all $C'_i \leftarrow C \setminus C_i$ do
7:        $C'_i \leftarrow C'_i \cup c_{ij}$
8:        if $\kappa(C') \geq \kappa(C)$ then
9:           return $C'$
10:   end if
11: end for
12: end for
13: end for
14: return $C$

vertex pair $(v_i, v_j) \in V$ if any cluster $C_i \in C$ exists on either vertices’ first neighbor both vertices are assigned to that cluster. Otherwise both vertices are made head of two new clusters $C_i, C_{i+1} \in C$.

4.2.2 Greedy Algorithm

A Greedy Algorithm always makes the choice that looks best at the moment. That means, it makes a locally optimal choice in the goal that this choice will lead a globally optimal solution. We present two variations of greedy algorithms in this thesis. Greedy algorithm for this problem not always find optimal solution, but it can be the initial seed to find better solution, as is used for our Local Search algorithm, Section 6.

Algorithm 4 presents the first variation of greedy based heuristics that have been applied. This algorithm selects each of vertices $v_i \in V$ in descending order of their nodal degree. A vertex is assigned to its adjacent neighboring cluster. If multiple adjacent neighboring clusters exists, than assign to the first one discovered. If no adjacent neighboring cluster is exist that vertex will be made the head
of a new cluster.

Algorithm 5 presents the second variation of greedy heuristics. Greedy-2 generates scalable number of clusters. This algorithm sort all the vertices based on the nodal degree and picks each of vertices \( v_i \in V \) in descending order. After selecting one vertex, we allocate its adjacent vertices including itself to a new cluster.

### 4.2.3 Local Search Algorithm

Local search is a heuristic method for solving computationally hard optimization problems[53]. Local search can be used on problems that have a number of candidate solutions to find the optimal solution. Local search algorithms move from solution to solution in the space of search space by applying local changes. Local search starts with a candidate solution, and looks for its neighboring solutions to find a better one. As soon a better neighboring solution is found, the search moves to that solution. This algorithm will be executed until the final optimal solution is found.

Algorithm 6 presents a local search heuristic for solving the design migration problem. This algorithm takes a Call Graph \( G = (V, E) \) as the search space. Local Search algorithm should have an initial solution seed \( C_{init} \) to start with neighbor solution. Algorithm 4 generates best result among all onter MonteCarlo and Greedy heuristics, solution of algorithm 4 has been taken as an initial solution \( C_{init} \) for our local search algorithm. In one step the algorithm searches over its neighbors in the solution space, and moves to a neighboring solution whenever it finds a better one, solution has been updated by the new one \( C \leftarrow C_{newLocOpt} \). This searching process stops when no update made over four round of loop.

Searching neighbor solution is executed by Algorithm 7 where a Solution \( C \) is taken as input. Every clusters \( C_i \in C' \) has been updated where \( C \) is a temporal solution \( C' \leftarrow C \). The solution update will be executed when the value of Kal-
Index (Eq.4.1) has been improved. The UpdateSolution function return the new neighbor candidate solution when $\kappa(C') \geq \kappa(C)$.

4.3 Summary

In this chapter we have described the mathematical formulation of the problem where we introduced new matrix to numerically measure software modularity. Several heuristic algorithms have been described in this chapter. We presented three variations of Monte Carlo Algorithms in Section 4.2.1, two variations of Greedy Algorithms in Section 4.2.2, and a Local Search based heuristic algorithm in Section 4.2.3.
Chapter 5
Experimental Results

This chapter presents the details of the experimental data that we have used numeric results and comparative study of the results obtained. Section 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, and 5.7 presents the Environmental Setup, Dataset Description, Execution Runtime, Monte Carlo algorithm results, Greedy algorithm results, Local Search algorithm results and all of their comparative results respectively.

5.1 Environmental Setup

Algorithms presented in Section 4.2 have been implemented using C++ programming language and the machine configuration is Operating System: Linux Mint15, Platform: 32bit, Processor Model: Intel Core-i3, Processor Speed: 2.20 GHz, Cache Memory: 6 MB, and RAM: 4 GB.

5.2 Dataset Description

We have reported experimental results on 6 problem instances. Instance BFT and RBIo [27] have been generated from two scientific research softwares, and the other instances Synthetic7, Synthetic71, Synthetic100 and Synthetic166 are synthetically generated. Table (5.2) presents the details of the dataset.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Vertices</th>
<th>Number of edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFT.tlp</td>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>RBIo.tlp</td>
<td>61</td>
<td>372</td>
</tr>
<tr>
<td>Synthetic7</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Synthetic71</td>
<td>71</td>
<td>160</td>
</tr>
<tr>
<td>Synthetic100</td>
<td>100</td>
<td>329</td>
</tr>
<tr>
<td>Synthetic166</td>
<td>166</td>
<td>450</td>
</tr>
</tbody>
</table>
5.3 Runtime Report

Table 5.2 reports the computational running time of different algorithms in microsecond. MC algorithms were executed 1000 times on each of the instances and the total execution times have been presented. The running times depend on the density of vertices and edges of the instance. According to instance BFT MC-3 is 34.66% faster in average than other MC algorithms. Greedy algorithms, since they have been executed only once, convincingly outperformed MC algorithms. In terms of computational time Greedy-2 is 0.59%, and 12.20% faster than Greedy-1 and Local Search receptively. Though Local Search takes more time for execution, but it produces better results than all other proposed heuristics algorithms.

<table>
<thead>
<tr>
<th>Data</th>
<th>MC1</th>
<th>MC2</th>
<th>MC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFT</td>
<td>15152</td>
<td>12316</td>
<td>10855</td>
</tr>
<tr>
<td>RBIo</td>
<td>1304418</td>
<td>1510649</td>
<td>1273883</td>
</tr>
<tr>
<td>Synthetic7</td>
<td>12747</td>
<td>8248</td>
<td>19793</td>
</tr>
<tr>
<td>Synthetic71</td>
<td>1197632</td>
<td>1432783</td>
<td>1986520</td>
</tr>
<tr>
<td>Synthetic100</td>
<td>3652895</td>
<td>3628750</td>
<td>3182150</td>
</tr>
<tr>
<td>Synthetic166</td>
<td>5729100</td>
<td>6100505</td>
<td>5617930</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
<th>Greedy-1</th>
<th>Greedy-2</th>
<th>Local Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFT</td>
<td>193</td>
<td>169</td>
<td>10532</td>
</tr>
<tr>
<td>RBIo</td>
<td>42238</td>
<td>23232</td>
<td>1710620</td>
</tr>
<tr>
<td>Synthetic7</td>
<td>56</td>
<td>50</td>
<td>13050</td>
</tr>
<tr>
<td>Synthetic71</td>
<td>40235</td>
<td>21730</td>
<td>2153502</td>
</tr>
<tr>
<td>Synthetic100</td>
<td>53210</td>
<td>42150</td>
<td>5495170</td>
</tr>
<tr>
<td>Synthetic166</td>
<td>67290</td>
<td>57945</td>
<td>8210395</td>
</tr>
</tbody>
</table>

5.4 Result of Monte Carlo Algorithm

The worst case, best case, average case, and median results of Monte Carlo algorithms in terms of CC ($\Psi$), CPL ($\chi$) and Kal-index ($\kappa$) has been presented in section 5.4.1 - 5.4.6 respectively. Output results are displayed using boxplot diagram where every box focuses the majority portion of the proposed result. The
solution value does not vary significantly among all the instance run except MC-1 algorithm.

5.4.1 Dataset BFT

The results of Monte Carlo algorithms for instance BFT are presented in Figure 5.1(a), 5.1(b), 5.1(c) present CC, CPL and Kal index respectively. For instance BFT the average and best result of (Ψ), (χ) and (κ) are (0.22194, 0.458333), (1.642, 0.7), and (5, 17) respectively. The largest and shortest running times of this instance are 19819 microseconds and 11855 microseconds respectively.

Figure 5.1: Monte Carlo Algorithms on BFT
5.4.2 Dataset RBIo

Figure 5.2 presents the results of all MC algorithms for instance RBIo. Sub-figure 5.2(a), 5.2(b), and 5.2(c) present CC, CPL and Kal index respectively. Output results are displayed using boxplot diagram where every box focuses the majority portion of the proposed result. For instance RBIo the average and best result of (Ψ), (χ) and (κ) are (0.21174, 0.478623), (1.768, 0.774771), and (5, 93) respectively. The largest and shortest running times of this instance are 1510649 microseconds and 1304418 microseconds respectively.

(a) Clustering Coefficient (Ψ)  
(b) Characteristics Path Length (χ)  
(c) KAL index (κ)

Figure 5.2: Monte Carlo Algorithms on RBIo
5.4.3 Dataset Synthetic7

Figure 5.3 presents the results of all MC algorithms for instance Synthetic7. Sub-figure 5.3(a), 5.3(b), and 5.3(c) present CC, CPL and Kal index respectively. For instance Synthetic7, the average and best result of $\Psi$, $\chi$, and $\kappa$ are $(0.81077, 0.8666)$, $(1.0168, 0.85)$, and $(4, 7)$ respectively. The largest and shortest running times of this instance are 19793 microseconds and 8248 microseconds respectively.

(a) Clustering Coefficient($\Psi$)  
(b) Characteristics Path Length($\chi$)  
(c) KAL index ($\kappa$)

Figure 5.3: Monte Carlo Algorithms on Synthetic7
5.4.4 Dataset Synthetic71

Figure 5.4 presents the results of all MC algorithms for instance Synthetic71. Sub-figure (a),(b),(c) present CC, CPL and Kal index respectively. For instance Synthetic71 the average and best result of (Ψ), (χ) and (κ) are (0.23058, 0.262826), (2.7168, 1.24348), and (47, 95) respectively. The largest and shortest running times of this instance are 1986520 microseconds and 1197632 microseconds respectively.

(a) Clustering Coefficient(Ψ)  
(b) Characteristics Path Length(χ)  
(c) KAL index (κ)  

Figure 5.4: Monte Carlo Algorithms on Synthetic71
5.4.5 Dataset Synthetic100

Figure 5.5 presents the results of all MC algorithms for instance Synthetic100. Sub-figure (a),(b),(c) present CC, CPL and Kal index respectively. For instance Synthetic100 the average and best result of $\Psi$, $\chi$ and $\kappa$ are (0.056607, 0.069126), (3.6188, 1.0329), and (-65, 30) respectively. The largest and shortest running times of this instance are 3652895 microseconds and 3182150 microseconds respectively.

![Box plots of Clustering Coefficient, Characteristics Path Length, and KAL index for Synthetic100](image)

Figure 5.5: Monte Carlo Algorithms on Synthetic100


### 5.4.6 Dataset Synthetic166

Figure 5.6 presents the results of all MC algorithms for instance *Synthetic166*; subsection (a),(b),(c) present CC, CPL and Kal index respectively. For instance *Synthetic166* the average and best result of (Ψ), (χ) and (κ) are 0.3215, 0.386517), (3.7154, 1.1102), and (81, 209) respectively. The largest and shortest running times of this instance are 6100505 microseconds and 5617930 microseconds respectively.

![Box plots of clustering coefficients, characteristic path lengths, and KAL indices](image)

(a) Clustering Coefficient (Ψ)  
(b) Characteristic Path Length (χ)  
(c) KAL index (κ)

Figure 5.6: Monte Carlo Algorithms on *Synthetic166*
5.5 Result of Greedy Algorithm

Greedy algorithms have been executed single time over a specific dataset. This algorithms find the locally optimal solution of design migration problem. We developed two variations of greedy algorithm to find desired solution. Each of variation produces better Clustering Coefficient ($\Psi$), Characteristics Path Length ($\chi$) and Kal-Index ($\kappa$) in terms of Monte Carlo algorithms. Table 5.3, 5.4, and 5.5 present the $\Psi$, $\chi$, and $\kappa$ of every dataset respectively.

The solution index of Greedy-1 and Greedy-2 may vary from instance to instance because of their data connectivity characteristics. Though Greedy-1 algorithm produces better result that Greedy-2 algorithm.

<table>
<thead>
<tr>
<th>BFT</th>
<th>RBIo</th>
<th>Synthetic7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy-1</td>
<td>0.324074</td>
<td>0.280905</td>
</tr>
<tr>
<td>Greedy-2</td>
<td>0.3925</td>
<td>0.219878</td>
</tr>
<tr>
<td>Synthetic71</td>
<td>0.282846</td>
<td>0.0468427</td>
</tr>
<tr>
<td>Greedy-1</td>
<td>0.198498</td>
<td>0.0408211</td>
</tr>
<tr>
<td>Greedy-2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3: Clustering Coefficient ($\Psi$) of Greedy-1 and Greedy-2

<table>
<thead>
<tr>
<th>BFT</th>
<th>RBIo</th>
<th>Synthetic7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy-1</td>
<td>0.98666</td>
<td>1.34956</td>
</tr>
<tr>
<td>Greedy-2</td>
<td>0.36667</td>
<td>0.736625</td>
</tr>
<tr>
<td>Synthetic71</td>
<td>1.62418</td>
<td>1.84989</td>
</tr>
<tr>
<td>Greedy-1</td>
<td>1.12732</td>
<td>1.06989</td>
</tr>
<tr>
<td>Greedy-2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4: Characteristics Path Length ($\chi$) of Greedy-1 and Greedy-2

<table>
<thead>
<tr>
<th>BFT</th>
<th>RBIo</th>
<th>Synthetic7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy-1</td>
<td>11</td>
<td>115</td>
</tr>
<tr>
<td>Greedy-2</td>
<td>7</td>
<td>49</td>
</tr>
<tr>
<td>Synthetic71</td>
<td>59</td>
<td>47</td>
</tr>
<tr>
<td>Greedy-1</td>
<td>57</td>
<td>51</td>
</tr>
<tr>
<td>Greedy-2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.5: Kal-Index ($\kappa$) of Greedy-1 and Greedy-2
5.6 Result of Local Search Algorithm

Local Search algorithm starts from an initial solution and find the better one. Since algorithm 5 produces best Kal(κ)-index, solution produced by Algorithm 5 is used as the initial solution seed to Local Search.

5.6.1 Dataset BFT

For dataset BFT initial solution Kal(κ)-index is 7 and the final Kal(κ)-index produced by Local Search is 13. The improvement of every iteration is plotted in Figure 5.7 and all 13 iterations of Kal(κ) results are 7, 7, 8, 9, 10, 11, 12, 13, 13, 13, 13, 13 respectively.

![Figure 5.7: Improvement of Kal(κ)-index using Local Search for dataset BFT](image)

Figure 5.7: Improvement of Kal(κ)-index using Local Search for dataset BFT
5.6.2 Dataset RBIo

Figure 5.8 presents the improvement of Kal(κ)-index for dataset RBIo using Local Search. Output of Algorithm 5 has been taken as initial solution seed and the κ of initial solution for dataset RBIo is 91. It takes 17 iteration to provide a optimal result. This execution has been stopped when no updates in last 4 iteration. The 17 iteration Kal(κ) results are 91, 93, 95, 97, 99, 103, 107, 111, 111, 113, 115, 119, 123, 123, 123, 123, 123 respectively.

![Figure 5.8: Improvement of Kal(κ)-index using Local Search for RBIo](image)

5.6.3 Dataset Synthetic71

Figure 5.9 presents the improvement of Kal(κ)-index for dataset Synthetic71 using Local Search. Output of Algorithm Greedy-1 has been taken as initial solution and the κ of initial solution for dataset Synthetic71 is 55 and the final optimal solution Kal(κ)-index is 145. It takes 48 iteration to provide the optimal solution. This execution has been stopped when no updates in last 4 iteration. The 48

5.6.4 Dataset Synthetic100

Figure 5.10 presents the improvement of Kal(κ)-index for dataset Synthetic100. The initial solution for this dataset is 55 and the final optimal solution Kal(κ)-index is 145. It takes 48 iteration to provide the optimal solution. This execution has been stopped when no updates in last 4 iteration. The 48 iteration Kal(κ) results are 55, 55, 57, 59, 59, 61, 61, 65, 65, 69, 73, 77, 77, 79, 79, 79, 81, 81, 85, 85, 87, 91, 95, 95, 99, 99, 105, 109, 111, 113, 113, 121, 121, 121, 129, 131, 131, 131, 135, 139, 145, 145, 145, 145, 145 respectively.
Figure 5.10: Improvement of Kal(\(\kappa\))-index using Local Search for Synthetic100

5.6.5 Dataset Synthetic166

Figure 5.11: Improvement of Kal(\(\kappa\))-index using Local Search for Synthetic166

The Kal(\(\kappa\))-indexes for dataset Synthetic166 have been presented by Figure
5.11 which are the product of every local search iteration. The Kal($\kappa$)-index of initial solution for this dataset is 255 and the final optimal solution Kal($\kappa$)-index is 353. It takes 30 iteration to provide the optimal solution. This execution has been stopped when no updates in last 4 iteration. The 30 iteration Kal($\kappa$) results are 5255, 265, 271, 271, 273, 279, 285, 291, 291, 291, 295, 297, 303, 305, 309, 315, 315, 319, 321, 329, 331, 335, 339, 341, 345, 350, 353, 353, 353, 353 respectively.

5.7 Comparative Result

The Comparative results between Monte Carlo, Greedy and Local Search algorithms on the experimental instances are given in Table 5.6 - Table 5.11 which are prepared from instance BFT, RBIo, Synthetic7, Synthetic71, Synthetic100 and Synthetic166 respectively. Each table presents the values of CCO($\Psi$), CPL($\chi$) and Kal($\kappa$) of every heuristics algorithms. Those tables proves that the proposed Local Search algorithm are 22.8%, 18.02%, and 135.73% better than proposed Monte Carlo algorithms and 8.25%, 11.32%, and 87.15% in terms of CC ($\Psi$), CPL ($\chi$) and Kal-index ($\kappa$) respectively.

To give a visual aid for results presented in Table 5.6 - Table 5.11, we provide comparison figures which has been attached as appendix A.

Table 5.6: MC-1 - MC-3 and Greedy-1-Greedy-2 and Local Search Algorithms on BFT

<table>
<thead>
<tr>
<th>Alg Name</th>
<th>CC Average</th>
<th>Best</th>
<th>CPL Average</th>
<th>Best</th>
<th>KL Average</th>
<th>Best</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC1</td>
<td>0.23542</td>
<td>0.435185</td>
<td>3.081011</td>
<td>1.76746</td>
<td>-5</td>
<td>5</td>
</tr>
<tr>
<td>MC2</td>
<td>0.237388</td>
<td>0.419753</td>
<td>1.22433</td>
<td>0.75</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>MC3</td>
<td>0.22513</td>
<td>0.366667</td>
<td>1.18216</td>
<td>0.75</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>Greedy-1</td>
<td>0.324074</td>
<td></td>
<td>0.98666</td>
<td></td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Greedy-2</td>
<td>0.3925</td>
<td></td>
<td>0.36667</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Local Search</td>
<td>0.3659722</td>
<td></td>
<td>0.601389</td>
<td></td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Alg Name</td>
<td>CC Average</td>
<td>CC Best</td>
<td>CPL Average</td>
<td>CPL Best</td>
<td>KL Average</td>
<td>KL Best</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>----------</td>
<td>-------------</td>
<td>-----------</td>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>MC1</td>
<td>0.3321</td>
<td>0.406779</td>
<td>3.0704</td>
<td>2.53297</td>
<td>-98</td>
<td>-81</td>
</tr>
<tr>
<td>MC2</td>
<td>0.3189</td>
<td>0.447633</td>
<td>1.0818</td>
<td>0.774771</td>
<td>3</td>
<td>91</td>
</tr>
<tr>
<td>MC3</td>
<td>0.3393</td>
<td>0.478623</td>
<td>1.0481</td>
<td>0.7</td>
<td>-3</td>
<td>93</td>
</tr>
<tr>
<td>Greedy-1</td>
<td>0.280905</td>
<td></td>
<td>1.34956</td>
<td></td>
<td></td>
<td>115</td>
</tr>
<tr>
<td>Greedy-2</td>
<td>0.219878</td>
<td></td>
<td>0.736625</td>
<td></td>
<td></td>
<td>49</td>
</tr>
<tr>
<td>Local Search</td>
<td>0.2965245</td>
<td></td>
<td>0.846941</td>
<td></td>
<td></td>
<td>123</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alg Name</th>
<th>CC Average</th>
<th>CC Best</th>
<th>CPL Average</th>
<th>CPL Best</th>
<th>KL Average</th>
<th>KL Best</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC1</td>
<td>0.7693</td>
<td>0.8666</td>
<td>1.6265</td>
<td>0.85</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>MC2</td>
<td>0.81077</td>
<td>0.8666</td>
<td>1.1078</td>
<td>0.85</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>MC3</td>
<td>0.80555</td>
<td>0.8666</td>
<td>1.10483</td>
<td>0.85</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Greedy-1</td>
<td>0.777</td>
<td>1.8</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Greedy-2</td>
<td>0.8333</td>
<td></td>
<td>1.16666</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Local Search</td>
<td>0.87777</td>
<td></td>
<td>1.06666</td>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alg Name</th>
<th>CC Average</th>
<th>CC Best</th>
<th>CPL Average</th>
<th>CPL Best</th>
<th>KL Average</th>
<th>KL Best</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC1</td>
<td>0.254821</td>
<td>0.262826</td>
<td>5.90161</td>
<td>5.6451</td>
<td>-115</td>
<td>-103</td>
</tr>
<tr>
<td>MC2</td>
<td>0.209488</td>
<td>0.23058</td>
<td>1.970065</td>
<td>1.24348</td>
<td>49</td>
<td>95</td>
</tr>
<tr>
<td>MC3</td>
<td>0.207768</td>
<td>0.208271</td>
<td>1.54241</td>
<td>1.38957</td>
<td>57</td>
<td>83</td>
</tr>
<tr>
<td>Greedy-1</td>
<td>0.282846</td>
<td></td>
<td>1.62418</td>
<td></td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>Greedy-2</td>
<td>0.198498</td>
<td></td>
<td>1.12732</td>
<td></td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Local Search</td>
<td>0.133939</td>
<td></td>
<td>1.02851</td>
<td></td>
<td>145</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alg Name</th>
<th>CC Average</th>
<th>CC Best</th>
<th>CPL Average</th>
<th>CPL Best</th>
<th>KL Average</th>
<th>KL Best</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC1</td>
<td>0.056622</td>
<td>0.058508</td>
<td>9.81</td>
<td>6.498</td>
<td>-227</td>
<td>-250</td>
</tr>
<tr>
<td>MC2</td>
<td>0.046623</td>
<td>0.056607</td>
<td>1.3172</td>
<td>1.40805</td>
<td>-67</td>
<td>-33</td>
</tr>
<tr>
<td>MC3</td>
<td>0.058349</td>
<td>0.069126</td>
<td>1.21503</td>
<td>1.0329</td>
<td>17</td>
<td>30</td>
</tr>
<tr>
<td>Greedy-1</td>
<td>0.0468427</td>
<td></td>
<td>1.84989</td>
<td></td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>Greedy-2</td>
<td>0.040821</td>
<td></td>
<td>1.06989</td>
<td></td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>Local Search</td>
<td>0.0242791</td>
<td></td>
<td>1.919457</td>
<td></td>
<td>151</td>
<td></td>
</tr>
</tbody>
</table>
Table 5.11: MC-1 - MC-3, Greedy-1, Greedy-2 and Local Search Algorithms on Synthetic

<table>
<thead>
<tr>
<th>Alg Name</th>
<th>CC Average</th>
<th>Best</th>
<th>CPL Average</th>
<th>Best</th>
<th>KL Average</th>
<th>Best</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC1</td>
<td>0.2693</td>
<td>0.34851</td>
<td>8.57</td>
<td>7.85</td>
<td>-379</td>
<td>-330</td>
</tr>
<tr>
<td>MC2</td>
<td>0.3215</td>
<td>0.3571</td>
<td>1.4578</td>
<td>1.1584</td>
<td>82</td>
<td>209</td>
</tr>
<tr>
<td>MC3</td>
<td>0.34555</td>
<td>0.386517</td>
<td>1.14083</td>
<td>1.1102</td>
<td>80</td>
<td>170</td>
</tr>
<tr>
<td>Greedy-1</td>
<td>0.34362</td>
<td></td>
<td>1.91511</td>
<td></td>
<td>309</td>
<td></td>
</tr>
<tr>
<td>Greedy-2</td>
<td>0.3035</td>
<td></td>
<td>1.8512</td>
<td></td>
<td>257</td>
<td></td>
</tr>
<tr>
<td>Local Search</td>
<td>0.180797</td>
<td></td>
<td>1.87614</td>
<td></td>
<td>353</td>
<td></td>
</tr>
</tbody>
</table>

5.8 Summary

In this chapter we have described the results of different proposed algorithms. Local Search produces better results than the greedy algorithms. However, the expense of the improvement was the longer running time. Local Search works excellent in large solution space, though in a small solution space is may not produce massive improvement of solution.
Chapter 6

Conclusion

This thesis proposes and studies a design migration problem involving SP and OOP. We represent the problem as graph clustering integer programming model and establish its computational hardness. We use three matrices to numerically measure the quality of clustering. Amongst these three the Kal ($\kappa$)-index is a unique contribution of this thesis that can be used to numerically measure the modularity of software design.

We have also proposed two variations of greedy heuristics and three variations of Monte Carlo based heuristics to solve the problem and test those algorithms against real and synthetic data, the algorithms and test results have also been reported in [54]. We have also implemented a Local Search based solution space search heuristics to solve this problem, and it produced the best result among all our proposed algorithms.

The work presented in this thesis has the following areas of development: Firstly, modeling the SP code as a call graph does not consider the global variables and function parameters (as has been used by [35]), which could be useful intuitions for the design of classes. Secondly, approaches presented in this thesis do not address discovery of polymorphism and inheritance of classes. Thirdly, we have not considered run time call scenario, i.e. how many times a function calls the other.

Further research on the problem may be directed towards the using numerical optimization problem solver software to run the model presented in Section 4.1 to get benchmarks results. We intend to try different variations of Local Search by redesign the neighborhood structure used in the algorithm presented in this thesis. Meta-heuristics solution space search algorithm such as Variable Neighborhood Search (VNS) and Variable Neighborhood Decomposition Search (VNDS) may be
applied using the Local Search we have developed. The Kal(κ)-index we have proposed is the first proposed numeric matrix to measure software modularity, this may be verified on existing software which has been designed Object Oriented Architecture.
Bibliography


[54] S. Siddik, A. U. Gias, and S. M. Khaled, “Optimizing software design migration from structured programming to object oriented paradigm,” in 16th International Conference on Computer and Information Technology (ICCIT 2013), (Khulna University, Khulna, Bangladesh), IEEE, December 2013. Accepted.
Appendix A

Comparison Chart on Experimental Results

Figure A.1, A.2, A.3, A.4, A.5, and A.6 presents the best and average result of MC algorithms in contrast to Greedy algorithms and Local Search algorithm. We see that the Local Search algorithms produce better result than the average performances of MC algorithms, and are almost closest of MC algorithms best results. Sign (\(\ast\)), (\(x\)), and (\(+\)) denote the average result of MC, best result of MC and optimal result of Greedy algorithms, and Local Search Algorithm respectively.

Name of algorithms and their corresponding results have been presented by X axis and Y axis respectively. For the variations of Monte Carlo algorithm MC-1, MC-2, and MC-3 plotted two point (\(\ast\)), and (\(x\)) which present average value and best value among all execution. Fourth and Fifth point of X axis present the Greedy-1 and Greedy-2 algorithm. Each greedy algorithm plotted the single result using (\(+\)) sign. Last and the sixth point of X axis presents the Local Search result using (\(+\)) sign. By analyzing all the result we got that the value of Local Search produces best result among all other proposed heuristic algorithms.
Figure A.1: Heuristics Algorithms Results Comparison on $BFT$
Figure A.2: Heuristics Algorithms Results Comparison on RBIo
Figure A.3: Huristics Algorithms Results Comparison on Synthetic
Figure A.4: Heuristics Algorithms Results Comparison on *Synthetic71*
Figure A.5: Heuristics Algorithms Results Comparison on *Synthetic100*
Figure A.6: Huristics Algorithms Results Comparison on Synthetic166
Appendix B

Monte Carlo Algorithms Result of 100 Execution

This chapter presents the best and average result of all proposed Monte Carlo algorithms on every execution. Each Monte Carlo algorithm has been executing 100 times and all the results of Clustering Coefficient ($\Psi$), Characteristics Path Length ($\chi$), and KAL index ($\kappa$) have been mentioned here. The horizontal line denotes the average results and the ($x$) points present the execution results.

Name of algorithms and their corresponding results have been denoted by X axis and Y axis respectively. Maximum results of Clustering Coefficient ($\Psi$) and KAL index ($\kappa$), and the minimal number of Characteristics Path Length ($\chi$) lead the better solution.
Figure B.1: Clustering Coefficient(Ψ) on BFT
Figure B.2: Characteristics Path Length($\chi$) of BFT
(a) KAL index ($\kappa$) MC-1

(b) KAL index ($\kappa$) MC-2

(c) KAL index ($\kappa$) MC-3

Figure B.3: KAL index ($\kappa$) of BFT
(a) Clustering Coefficient ($\Psi$) MC-1

(b) Clustering Coefficient ($\Psi$) MC-2

(c) Clustering Coefficient ($\Psi$) MC-3

Figure B.4: Clustering Coefficient ($\Psi$) on RBIo
Figure B.5: Characteristics Path Length(χ) of RBIo
Figure B.6: KAL index (κ) of RBLo
Figure B.7: Clustering Coefficient(Ψ) on Synthetic71
Figure B.8: Characteristic Path Length($\chi$) of Synthetic71
Figure B.9: KAL index ($\kappa$) of Synthetic71
Figure B.10: Clustering Coefficient($\Psi$) on Synthetic100
Figure B.11: Characteristics Path Length($\chi$) of Synthetic100
Figure B.12: KAL index ($\kappa$) of Synthetic100
Figure B.13: Clustering Coefficient(Ψ) on Synthetic166
Figure B.14: Characteristics Path Length($\chi$) of Synthetic166
(a) KAL index ($\kappa$) MC-1

(b) KAL index ($\kappa$) MC-2

(c) KAL index ($\kappa$) MC-3

Figure B.15: KAL index ($\kappa$) of Synthetic166