

**CLUSTERING SOFTWARE SYSTEMS TO IDENTIFY  
SUBSYSTEM STRUCTURES USING KNOWLEDGEBASE**

**BY**

**MD. NASIM ADNAN**

A thesis submitted to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering




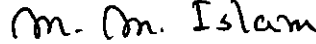
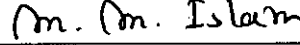
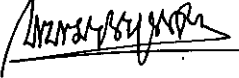

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY  
DHAKA, BANGLADESH**

**JANUARY 2010**

## CERTIFICATE OF APPROVAL

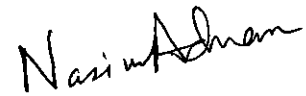
The thesis titled “Clustering Software Systems to Identify Subsystem Structures using Knowledgebase” submitted by Md. Nasim Adnan, Roll No: 040505035P, Session: April, 2005 to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirement for the degree of **Master of Science in Computer Science and Engineering** on January 14, 2010.

### Board of Examiners

- |    |   |                          |
|----|---|--------------------------|
| 1. | <br>Dr. Md. Mostofa Akbar<br>Associate Professor<br>Department of CSE, BUET, Dhaka.  | Chairman<br>(Supervisor) |
| 2. | <br>Dr. Md. Monirul Islam<br>Head<br>Department of CSE, BUET, Dhaka.   | Member<br>(Ex-officio)   |
| 3. | <br>Dr. Md. Monirul Islam<br>Professor<br>Department of CSE, BUET, Dhaka.  | Member                   |
| 4. | <br>Dr. M. Kaykobad<br>Professor<br>Department of CSE, BUET, Dhaka.  | Member                   |
| 5. | <br>Dr. Md. Zahidur Rahman<br>Professor<br>Department of Computer Science & Engg.<br>Jahangir Nagar University, Savar, Dhaka | Member<br>(External)     |

## **CANDIDATE'S DECLARATION**

This is to certify that the work entitled "**Clustering Software Systems to Identify Subsystem Structures using Knowledgebase**" is the outcome of the research carried out by me under the supervision of Dr. Md. Mostofa Akbar in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka-1000. It is also declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.



---

Md. Nasim Adnan

Candidate

## Table of Contents

|  |             |
|--|-------------|
| <b>Board of Examiners</b> .....  | <b>ii</b>   |
| <b>Candidate's Declaration</b> .....   | <b>iii</b>  |
| <b>Table of Contents</b> .....   | <b>iv</b>   |
| <b>List of Figures</b> .....   | <b>vii</b>  |
| <b>List of Tables</b> .....  | <b>ix</b>   |
| <b>Glossary of Terms</b> .....   | <b>x</b>    |
| <b>Acknowledgement</b> .....   | <b>xi</b>   |
| <b>ABSTRACT</b> .....  | <b>xiii</b> |
| <b>Chapter 1: Introduction</b> .....   | <b>1</b>    |
| 1.1 Motivation .....   | 1           |
| 1.2 Software Clustering Problem .....  | 2           |
| 1.3 Software Clustering as an Emerging Area of Research.....                         | 2           |
| 1.4 Issues in Software Clustering .....  | 2           |
| 1.5 Main Focus .....   | 3           |
| 1.6 Outline of the rest of the Thesis.....   | 3           |
| <b>Chapter 2: Literature Review and Related Works</b> .....                          | <b>5</b>    |
| 2.1 Major Components of Clustering Approaches .....                                  | 5           |
| 2.1.1 Similarity Measurement .....   | 5           |
| 2.1.2 Different Types of Clustering Algorithms .....                                 | 7           |
| 2.2 Existing Clustering Algorithms.....  | 9           |
| 2.3 Existing Software Clustering Algorithms .....                                    | 10          |
| 2.3.1 An Algorithm for Comprehension-driven clustering .....                         | 10          |
| 2.3.2 Information-Theoretic Software Clustering .....                                | 10          |
| 2.3.3 A Heuristic Approach to Solving the Software Clustering Problem .....          | 11          |
| 2.3.4 Reverse Engineering Software Architecture using Rough Clusters .....           | 13          |
| 2.3.5 Other Prominent Clustering Approaches.....                                     | 16          |
| 2.4 Observations.....  | 16          |
| 2.5 Research Challenges.....   | 17          |
| 2.6 Chapter Summary .....  | 18          |
| <b>Chapter 3: Software Clustering Using the Knowledgebase</b> .....                  | <b>19</b>   |
| 3.1 Problems with Existing Software Clustering Approaches.....                       | 19          |
| 3.1.1 Existing Similarity Measurement Techniques for Software Systems....            | 19          |
| 3.1.2 Our Solution to this Problem: Software Clustering using the Knowledgebase..... | 22          |

|                   |  |           |
|-------------------|--|-----------|
| 3.2               | The Knowledgebase .....  | 23        |
| 3.2.1             | Knowledgebase Development.....   | 23        |
| 3.2.2             | Knowledgebase Representation.....  | 24        |
| 3.3               | Similarity Measurement using the Knowledgebase .....                                       | 26        |
| 3.3.1             | Example of Similarity Measurement Computation.....   | 27        |
| 3.4               | Subsystem Identification using the Knowledgebase .....                                     | 28        |
| 3.5               | Software Clustering Using the Knowledgebase .....  | 29        |
| 3.6               | Pseudo Codes.....  | 31        |
| 3.7               | Comparison of the Proposed New Clustering Method over the Existing Methods.....            | 34        |
| 3.8               | Complexity Analysis .....  | 35        |
| 3.9               | Chapter Summary .....  | 37        |
| <b>Chapter 4:</b> | <b>Implementation of Clustering System .....</b>   | <b>38</b> |
| 4.1               | Key Modules in <i>BUET Cluster 1.0</i> .....   | 38        |
| 4.2               | Source Code Analysis and Creation of <i>MDG</i> .....                                      | 39        |
| 4.2.1             | Source Code Analysis using <i>Understand 2.0</i> .....                                     | 40        |
| 4.2.2             | Creation of the <i>MDG</i> .....   | 45        |
| 4.2.3             | Visualization of <i>MDG</i> using <i>Graphviz-win v2.16</i> .....                          | 47        |
| 4.3               | Generation of the Subsystems using <i>BUET Cluster 1.0</i> .....                           | 49        |
| 4.3.1             | Knowledgebase Development.....   | 49        |
| 4.3.2             | Knowledgebase Adaptation .....   | 51        |
| 4.3.3             | Cluster Generation.....  | 52        |
| 4.4               | Clustering software systems using <i>BUNCH</i> and <i>ACDC</i> .....                       | 53        |
| <b>Chapter 5:</b> | <b>Results with Comparison.....</b>  | <b>54</b> |
| 5.1               | Comparison Criteria .....  | 54        |
| 5.2               | MoJo Distance for Comparison of Clustering Quality.....                                    | 55        |
| 5.3               | Resultant Clusters of the Library Management System.....                                   | 56        |
| 5.3.1             | Results generated by <i>BUET Cluster 1.0</i> .....   | 56        |
| 5.3.4             | Results generated by <i>BUNCH</i> .....  | 58        |
| 5.3.5             | Result generated by <i>ACDC</i> .....  | 59        |
| 5.4               | Resultant Clusters of the Xfig .....   | 60        |
| 5.4.1             | Results generated by <i>BUET Cluster 1.0</i> .....   | 60        |
| 5.4.2             | Results generated by <i>BUNCH</i> .....  | 61        |
| 5.4.3             | Result generated by <i>ACDC</i> .....  | 62        |
| 5.5               | Comparison among the results of <i>BUET Cluster 1.0</i> , <i>BUNCH</i> and <i>ACDC</i> ... | 62        |
| 5.6               | Comparison with other Clustering Approaches.....   | 63        |

|                                      |           |
|--------------------------------------|-----------|
| <b>Chapter 6: Conclusions .....</b>  | <b>65</b> |
| 6.1 Major Contributions .....        | 65        |
| 6.2 Future Research Directions ..... | 66        |

## List of Figures

|   |    |
|---|----|
| Figure 2.1 Entities with features .....   | 6  |
| Figure 2.2 Example of partition sequence .....  | 8  |
| Figure 2.3 The dendrogram for the example partition sequence .....                          | 8  |
| Figure 2.4 A small compiler .....   | 12 |
| Figure 2.5 Output from <i>BUNCH</i> for the small compiler .....                            | 13 |
| Figure 2.6 Rough clusters .....   | 14 |
| Figure 3.1 Knowledgebase Development .....  | 24 |
| Figure 3.2 Simple relational Knowledgebase for similarity measurement.....                  | 26 |
| Figure 3.3 A typical example of two functions.....  | 27 |
| Figure 3.4 Architecture of the proposed clustering approach.....                            | 30 |
| Figure 3.5 Flow Chart of the proposed clustering process .....                              | 31 |
| Figure 4.1 Modules in clustering approaches .....   | 39 |
| Figure 4.2 Run of <i>Understand 2.0</i> .....   | 42 |
| Figure 4.3 Main window of <i>Understand 2.0</i> used to create the new project.....         | 41 |
| Figure 4.4 Source Code of Library Management System as input to <i>Understand 2.0</i> ..... | 41 |
| Figure 4.5 Analysis of the Source Code .....  | 42 |
| Figure 4.6 Intermediate file generation .....   | 42 |
| Figure 4.7 Run of <i>BUET Cluster 1.0</i> .....   | 45 |
| Figure 4.8 Parsing of <i>UnderstandLibrary.txt</i> .....                                    | 46 |
| Figure 4.9 Editor of Graphviz-win v2.16 .....   | 48 |
| Figure 4.10 <i>DOT</i> layout engine to view the <i>MDG</i> .....                           | 48 |
| Figure 4.11 <i>MDG.jpg</i> to view the <i>MDG</i> .....                                     | 49 |
| Figure 4.12 Knowledgebase Development.....  | 50 |
| Figure 4.13 Soul cluster identification.....  | 51 |
| Figure 4.14 Resultant subsystems.....   | 52 |
| Figure 4.15 Pathway for running <i>BUNCH</i> and <i>ACDC</i> .....                          | 53 |
| Figure 5.1 Gold Standard for Library Management System .....                                | 55 |
| Figure 5.2 Gold Standard for Xfig.....  | 56 |

|   |    |
|---|----|
| Figure 5.3 Resultant Clusters from <i>BUET Cluster 1.0 (with threshold 8)</i> ..... | 56 |
| Figure 5.4 Resultant Clusters from <i>BUET Cluster 1.0 (with threshold 9)</i> ..... | 57 |
| Figure 5.5 Result 1.....  | 58 |
| Figure 5.6 Result 2.....  | 58 |
| Figure 5.7 Result 3.....  | 59 |
| Figure 5.8 Resultant Clusters from <i>ACDC</i> .....                                | 59 |
| Figure 5.9 Resultant Clusters from <i>BUET Cluster 1.0</i> .....                    | 60 |
| Figure 5.10 Resultant Clusters from <i>BUNCH</i> .....                              | 61 |
| Figure 5.11 Resultant Clusters from <i>ACDC</i> .....                               | 62 |



## List of Tables

|   |    |
|---|----|
| Table 3.1 Entity with their associated features .....                                       | 20 |
| Table 3.2 Similarity table using Jaccard coefficient.....                                   | 20 |
| Table 3.3 Entity with their associated features .....                                       | 21 |
| Table 3.4 Similarity table using Weighted scheme .....                                      | 21 |
| Table 3.5 Facts associated with the Generic Types.....                                      | 25 |
| Table 3.6 Example of a part of the Similarity Matrix.....                                   | 28 |
| Table 4.1 Brief content of the Intermediate Representation .....                            | 43 |
| Table 5.1 Results from <i>BUET Cluster 1.0</i> for Xfig for different threshold values..... | 60 |
| Table 5.2 Results from <i>BUNCH</i> for Xfig .....  | 61 |
| Table 5.3 Comparison among <i>ACDC</i> , <i>BUNCH</i> and <i>BUET Cluster 1.0</i> .....     | 62 |
| Table 5.4 Characteristic observation.....   | 64 |

## Glossary of Terms

|       |   |
|-------|---|
| QT    | Quality Threshold                                     |
| ACDC  | Algorithm for Comprehension-Driven Clustering         |
| LIMBO | scaLable InforMation BOttleneck                       |
| AIB   | Agglomerative Information Bottleneck                  |
| SA    | Summary Artifact                                      |
| MQ    | Modularization Quality                                |
| BUCNH | Tool for Software Clustering using Heuristic Approach |
| ADG   | Artifact Dependency Graph                             |
| BCM   | Business-Concept Model                                |
| LT    | Lower Threshold                                       |
| HT    | Higher Threshold                                      |
| DBMS  | Database Management Systems                           |
| OS    | Operating System                                      |
| MDG   | Module Dependency Graph                               |
| IDE   | Interactive Development Environment                   |
| API   | Application Programming Interface                     |
| COM   | Component Object Model                                |

## Acknowledgement

I would like to express my deepest appreciation to my supervisor Dr. Md. Mostofa Akbar for his generous help and thoughtful suggestions. He acted as a mentor and gave me much of his valuable time. His innovating and thinking capability helped me a lot.

I want to thank all my colleagues of Bangladesh Bank to help me in different ways. I am especially thankful to my CSE, KU friend Mr. Mohammad Monoar Hossain for his generous help and guidance. Thanks to Mr. Kevin Groke for allowing me to register for an academic license for *Understand 2.0*.

I also want to thank the other members of my thesis committee: Dr. Md. Monirul Islam, Dr. M. Kaykobad and Dr. Md. Zahidur Rahman for their valuable suggestions.

I would like to express my gratitude to my parents Dr. Md. Nur-un-nabi and Mrs. Nasrin Akhtar for giving me support and love. I am also thankful to my brothers Mr. Md. Nahid Noor Tusher and Mr. Md. Nashid Farhad for their generous support. I express my thanks to my sister-in-law Miss. Zannatul Zakia Tumpa for her co-operation.

Finally, I want to thank my wife Mrs. Afroza Sultana who helped me a lot during the course of the thesis.

*To My Mother Nasrin Akhtar, Ph. D.*

*&*

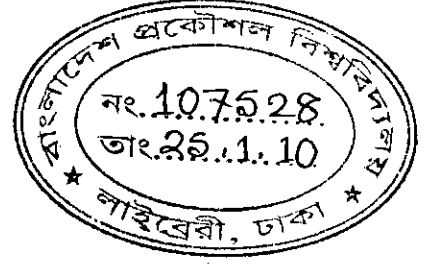
*Father Dr. Md. Nur-Un-Nabi*

## ABSTRACT

The structure of a software system deteriorates as a result of continuous maintenance activity. For the purpose of software reengineering or reverse engineering, often the software engineers get only the original source code as the most updated source of information due to lack of current documentation and limited or nonexistent availability of the original designers. The application of clustering techniques to the software systems aiming to discover the feature-oriented and meaningful subsystems helps the software engineers to understand the high-level features provided by those subsystems which is very essential for the purpose of software reengineering and reverse engineering. Continuous research is going on in the recent years –addressing different issues in the software clustering problem. Similarity measurement is the key to perform successful clustering. The similarity measurement criteria used in the existing clustering technique has the common drawback that they do not incorporate the diversity of software systems. Our approach introduces the use of the Knowledgebase which acts as the repository of information about the internal structure of the Generic types of the software systems to provide the guidelines on similarity measurement criteria and weights. The final clustering is done by integrating automatically generated subsystems with the known subsystems (provided by the Knowledgebase). Thus the new clustering technique is a semi automatic technique with the provision of tuning the results by the software engineers. In our research, we have developed a tool named *BUET Cluster 1.0* which implements our new clustering technique. This clustering tool has been evaluated by using a benchmark named *MoJo distance* for different well known software systems. The experimental results show that our approach generates more appropriate subsystems than the other existing clustering techniques and outperforms other clustering techniques in different dimensions of software clustering quality.

# Chapter 1

## Introduction



The structure of a software system deteriorates as a result of continuous maintenance activities. This problem is exacerbated by the fact that the documentation of the software system is rarely updated along with limited or nonexistent availability of the key developers. Often the original source code is the only available source of information and it is very time consuming to study it manually because of its sheer size and complexity. But software engineers need to understand the structure of the software systems to be able to make effective modifications and improvements. As a result, there is always a need to develop clustering techniques for understanding the structure of the software systems from the available source codes.

### 1.1 Motivation

Commercial or industrial software systems commonly have two properties that make the understanding of their structure a difficult task: they are large and complex. A typical system consists of thousands of entities (e.g., functions, procedures, classes, variables) and those are interconnected in intricate ways (e.g., function/procedure calls, inheritance relations, variable references).

Software Engineering textbooks advocate the use of documentation as an essential tool for describing a system's intended behavior and for capturing the system's structure. Clearly, in order to be useful to future software maintainers, a system's documentation must be updated with respect to any software changes. In practice, however, we often find that accurate and current design documentation does not exist. Because, once an understanding of the system structure is formed, it is difficult to maintain the documentation updated as the software systems tend to evolve during maintenance. This problem is exacerbated as the key designers and developers of the system are often no longer available for consultation. Often the designers have found a new job or are working on another project with deadlines that preclude them from providing advice to the current maintainers of the system.

In the absence of advice or current documentation about a system's structure, software engineers are left with very limited choices. Briefly, they can manually inspect the source code to develop a mental model of the system organization. This approach is often not practical because of the large number of dependencies between the source code components. So, an alternative comes forward to the software engineers is to use clustering techniques suitable for software systems to produce useful information about the system structure.

## 1.2 Software Clustering Problem

Clustering is the process of grouping together items or entities based on their properties or features. This grouping (clustering) reduces the amount of data that must be understood, which makes the original problem easier to understand. The application of clustering techniques in the area of software systems focus on recovering subsystems to understand the high-level features provided by those subsystems which is very much essential for the purpose of software reengineering. The input of the clustering process is the source code of the application.

## 1.3 Software Clustering as an Emerging Area of Research

Clustering has been applied in many disciplines like mathematics, social sciences, engineering, biology, astronomy, archaeology etc. where it is used to group together similar objects thus making large data sets more understandable and also in helping to find reasons for an entity's behavior.

The use of clustering in the arena of software systems is a relatively newer area of research (within the last 30 years). As the software systems exhibit some peculiarities in comparison to the subjects of other disciplines, it is required to tailor clustering techniques keeping in mind the type of software systems and the purpose of clustering. These situations have resulted the proliferation of new clustering techniques in the recent years.

## 1.4 Issues in Software Clustering

Clustering a software system into meaningful modules is very much important to make the clustering process efficient for software engineers. Following issues need to be addressed for successful clustering:

- *Cohesion and Coupling*: There is no guarantee that the developers of a legacy software system have followed software engineering principles such as high-cohesion and low-coupling. As a result, the validity of the clusters discovered following such principles, as well as the overall contribution of the obtained decomposition to the reverse engineering process, can be challenged.
- *Nonstructural Information*: Any type of nonstructural information such as timestamps, naming conventions, ownership information, comments etc. does not have the credibility to be used in software clustering approaches. Because, all these information may be partially present or totally absent in the source code of any software system. Also, it is not clear what types of nonstructural attributes are appropriate for inclusion in the software clustering approach.
- *Stability*: The results of the software clustering approaches should demonstrate stability. To be stable, the algorithm must produce software clustering results that do not vary widely from one version of a software system to the next (after being changed slightly). An approach which produces software clustering results that vary widely even though no major code restructuring occurred between versions is likely not to be reliable in the context of software clustering.

## 1.5 Main Focus

The main focus of software clustering is to discover valid feature-oriented subsystems which will really help the software engineers. In this thesis we do not develop the source code analyzer as there are available source code analyzers which are being used for other clustering technique. We also use off the shelf visualization tool to visualize the clusters to be produced. In this research we concentrate on the algorithm of clustering. A tool has been developed to demonstrate a practical clustering system using our algorithm. We also compare all the available clustering technique with our new technique.

## 1.6 Outline of the rest of the Thesis

The thesis is organized into six chapters as shown in below:

- Chapter 2 (*Literature Review and Related Works*): This chapter describes major components of software clustering. Other prominent approaches are also described here.



- Chapter 3 (*Software Clustering using the Knowledgebase*): This chapter describes our proposed software clustering approach along with block diagram, flow chart, pseudo codes and complexity analysis.
- Chapter 4 (*Implementation of Clustering System*): This chapter gives a detailed description of implementing our software clustering approach.
- Chapter 5 (*Results with Comparison*): This chapter presents the results generated by our approach and other available approaches. This chapter also contains a detailed comparison among them.
- Chapter 6 (*Conclusions*): This chapter concludes this thesis focusing on major contributions and future research scopes.

# Chapter 2

## Literature Review and Related Works

Software clustering problem has attracted many researchers in the last two decades. The importance that the software engineering community assigns to the software clustering problem is indicated by large number of publications on the subject that appear in many research articles on software engineering. Now, we introduce some software clustering terminologies which appear in the research articles more frequently. The elements to be clustered in the arena of software clustering such as “entities”, “functions”, “procedures” etc. are commonly referred to as software artifacts. Moreover, the terms “decomposition”, “subsystem generation”, “partition” and “clustering” will be used interchangeably to describe the output of software clustering algorithm.

### 2.1 Major Components of Clustering Approaches

Clustering approaches from every discipline are more or less similar in their composition. Every clustering approach has two major components namely – the similarity measurement and the clustering algorithm.

#### 2.1.1 Similarity Measurement

One of the first things that a clustering approach usually does is to decide on what grounds two objects will be judged to be similar. That is, one needs a measure that will decide which pair of objects are “more similar” than any other pair. The answer to this problem is: “similarity measures”. Similarity measures are designed in such a way that larger values indicate a stronger similarity between the objects.

Three types of similarity measures are typically employed in clustering applications [10]. Those are described as follows:

*Association coefficients:* Association coefficients are applied to calculate similarity when the features are binary. To illustrate how these coefficients are calculated, assume two entities  $E_1$  and  $E_2$ , represented by feature vectors indicating the presence or absence of a feature. The similarity between  $E_1$  and  $E_2$  can be compactly represented by a table as shown in Figure 2.1.

|       |   |       |     |
|-------|---|-------|-----|
|       |   | $E_2$ |     |
|       |   | 1     | 0   |
| $E_1$ | 1 | $a$   | $b$ |
|       | 0 | $c$   | $d$ |

*Fig. 2.1 Entities with features*

In the above table  $a$  represents the count of features present in both  $E_1$  and  $E_2$ ,  $b$  represents the total number of features present in  $E_1$  but absent in  $E_2$ ,  $c$  represents the total number of features present in  $E_2$  but absent in  $E_1$ , and  $d$  represents the number of features that are absent in both  $E_1$  and  $E_2$ . It is worth noting that in the software domain, typically  $d$  will be much larger than  $a, b, c$  since the feature vector associated with each entity is likely to be sparse. The following association coefficients can then be defined:

- Jaccard coefficient,  $J = a/(a+b+c)$
- Simple coefficient,  $S = (a+d)/(a+b+c+d)$
- Sorensen-Dice coefficient,  $SD = 2a/(2a+b+c)$
- Ellenberg measure (non-binary counterpart of Jaccard coefficient)

$$E = \frac{1/2 * Ma}{1/2 * (Ma + Mb + Mc)}, \text{ Here } M \text{ is defined as the magnitude of the variables.}$$

*Distance measures:* The distance measures calculate the dissimilarity between entities. The larger the distance, the lesser is the similarity between the entities. The measure is zero if and only if the entities have the same score on all features. Some popular distance measures are:

- Euclidean Distance,  $D(X, Y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$
- Canberra Distance,  $C(X, Y) = \sum_{i=1}^n |x_i - y_i| / (|x_i| + |y_i|)$
- Minkowski Distance,  $M(X, Y) = \left( \sum_{i=1}^n |x_i - y_i|^r \right)^{1/r}$

*Correlation Coefficients:* Correlation coefficients are used to correlate features. The well-known Pearson product moment correlation coefficient is given by:

$$\rho = \frac{\sum_{i=1}^n (x_i - \bar{x}_i)(y_i - \bar{y}_i)}{\sqrt{\sum_{i=1}^n (x_i - \bar{x}_i)^2 \sum_{i=1}^n (y_i - \bar{y}_i)^2}}$$

As the software artifacts are different in nature than the entities of other disciplines, similarity measures must be modified so that they can be applicable on the software artifacts.

### 2.1.2 Different Types of Clustering Algorithms

Once the similarity measure has been decided upon, an appropriate algorithm has to be chosen for the purpose of clustering. Clustering algorithms can be broadly divided into categories, namely partitional and hierarchical [18].

*Partitional algorithms:* Partitional algorithms usually work by starting with an initial partition and try to modify it in an attempt to optimize a criterion that represents the quality of a given partition. But the challenge that partitional algorithms face is the combinatorial explosion of the number of partitions. Even for a small number of objects the number of possible partitions is astronomical. For example, there are 34,105 partitions of ten objects into four clusters but this number explodes to approximately 11,259,666,000 if the number of objects is increased to 19.

The usual workaround to this problem is to start with an initial partition (chosen randomly or based on some heuristics) and attempt to optimize the chosen criterion by modifying that partition in an appropriate way. Therefore, the choice of initial partition is crucial for success of this algorithm.

*Hierarchical algorithms:* Hierarchical algorithms produce a nested sequence of partitions. At one end this sequence is the partition where each object is in a different cluster (we will call this partition ALL) and at the other end the partition where all are in the same cluster (we will call this partition ONE). At each step through this sequence two of the clusters are joined together. Figure 2.2 shows an example partition sequence for four objects A, B, C and D.

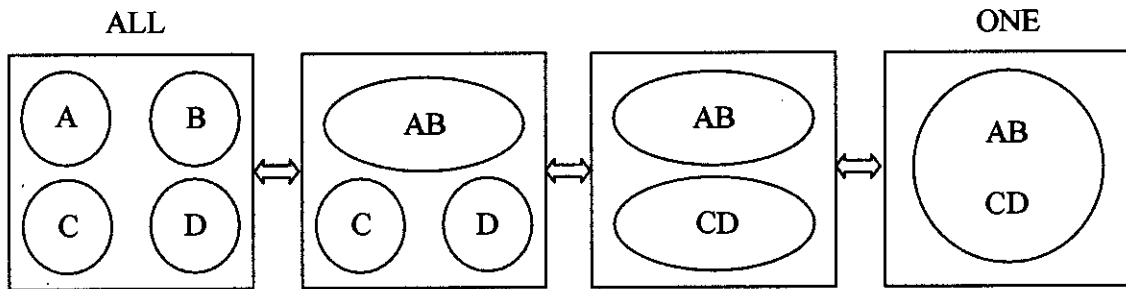


Fig. 2.2 Example of partition sequence

A common representation for a hierarchical structure is that of a *dendrogram*. Figure 2.3 shows the dendrogram for the of example partition sequence shown in Figure 2.2.

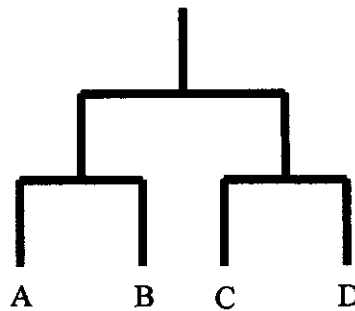


Fig. 2.3 The dendrogram for the example partition sequence

However, clustering algorithms need a “cut point” as it is not realistic to generate a single (ONE) cluster. Factors that influence the selection of a “cut point” are usually a priori knowledge on the expected structure or pre-chosen parameters such as the maximum number of clusters allowed or the maximum number of objects in a cluster.

Hierarchical algorithms are divided into two categories, agglomerative and divisive:

- *Agglomerative (bottom-up)*: These starts with partition ALL and iteratively join the most similar clusters based on the similarity measure.
- *Divisive (top-down)*: These start with partition ONE and try to iteratively split it until we reach partition ALL. Such algorithms suffer from excessive computational complexity as there is an exponential number of possible partitions at every step. This is the main reason why these algorithms are not very popular.

Regular clustering algorithms are generally used for data partitioning [12]. In the next subsection we present some renowned clustering algorithms.

## 2.2 Existing Clustering Algorithms

*k-means clustering:* *k*-means clustering [12], [22] is one type of partitional algorithm which aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest *mean*. The steps of the algorithm are:

- Choose the number of clusters, *k*.
- Randomly generate *k* clusters and determine the cluster centers, or directly generate *k* random points as cluster centers.
- Assign each point to the nearest cluster center.
- Recompute the new cluster centers.
- Repeat the two previous steps until some convergence criterion is met.

The complexity of *k*-means algorithm is  $O(knI)$ , where *I* is number of iterations.

*Quality Threshold clustering:* Quality Threshold (*QT*) clustering [22] is method of partitioning data invented for gene clustering. It requires more computing power than *k*-means, but does not require specifying the number of clusters a priori, and always returns the same result when run several times. In this algorithm:

- The user chooses a maximum diameter for clusters.
- Build a candidate cluster for each point by including the closest point, the next closest, and so on, until the diameter of the cluster surpasses the threshold.
- Save the candidate cluster with the most points as the first true cluster, and remove all points in the cluster from further consideration.
- Recurse with the reduced set of points.

*Monothetic divisive method:* This method of clustering [12] splits clusters using one variable at a time. This is a convenient (though restrictive) way to limit the number of possible patterns that must be examined. It has the attraction that the result is easily described by the dendrogram – the split at each node is defined in terms of just a single variable.

*Polythetic divisive method:* This method of clustering [12] makes splits on the basis of all the variables together. Any inter-cluster distance measure can be used. The difficulty

comes in deciding how to choose potential allocations to clusters. In one approach, objects are examined one at a time, and that one is selected for transfer from a main cluster to a subcluster that leads to the greatest improvement in the clustering score. In general, this method is computationally intensive and tends to be less widely used.

There are many types of clustering techniques like *probabilistic model-based clustering* [12], *spectral clustering* [22] and so on. All these cluster analysis techniques are very much data-driven, with relatively little formal model-building capability [12].

## 2.3 Existing Software Clustering Algorithms

Software clustering is different in nature from data clustering and relatively newer area of research. We have studied several software clustering approaches and now in this section we will present some prominent approaches.

### 2.3.1 An Algorithm for Comprehension-driven clustering

Tzerpos et. al. [1] proposed a pattern-based software clustering algorithm named Algorithm for Comprehension-Driven Clustering (*ACDC*) that performs the task of clustering in two stages. In the first stage, it creates a skeleton of the final decomposition by identifying subsystems that resemble established subsystem patterns of the original software system. Depending on the pattern used by the subsystems, they are given appropriate names. In the second stage, *ACDC* completes the decomposition by using an extended version of a technique known as Orphan Adoption. Orphan Adoption is an incremental clustering technique based on the assumption that the existing structure is well established. It attempts to place each newly introduced resource (called an orphan) in the subsystem that seems “more appropriate”. The complexity of this algorithm is NP complete.

### 2.3.2 Information-Theoretic Software Clustering

Here, Andritsos et. al. [2] introduced the scaLable InforMation BOttleneck (*LIMBO*) algorithm which is an improvement of the Agglomerative Information Bottleneck (*AIB*) algorithm [8] and capable of handling larger inputs. The aim of this algorithm is to use the *AIB* algorithm but on a smaller set of artifacts. In order to reduce the number of artifacts, it performs an initial phase, where artifacts are summarized in a newly created set of summaries that we call Summary Artifacts (*SA*).

*The LIMBO Clustering Algorithm:*

The LIMBO algorithm proceeds in four phases. These are –

*Phase 1 (Creation of the Summary Artifacts):* In this phase, original artifacts are read one by one. The first artifact  $a_1$  is converted into the summary artifact  $SA(\{a_1\})$ . For each subsequent artifact  $a_i$ , the algorithm computes its distance to each existing summary artifact. Next, we identify the summary artifact  $SA(S_{min})$  with the smallest distance to  $a_i$ . If this distance is smaller than a predefined threshold, then  $SA(S_{min})$  is replaced by a new summary artifact  $SA(S_{min} \cup \{a_i\})$ . The complexity of this phase is  $O(qEn \log_E n)$  with I/O cost  $O(n)$ .

*Phase 2 (Application of the AIB algorithm):* In this phase, the algorithm employs the AIB algorithm to cluster the set of SAs. This phase creates many clusterings of the summary artifacts, one for every value between 2 and  $\|S\|$  (number of all summary artifacts). The time for this phase depends upon the number of all summary artifacts, which can be much smaller than all software artifacts. The complexity of this phase is  $O(\|S\|^2 \log \|S\|)$ .

*Phase 3 (Associating original artifacts with clusters):* Phase 2 produces  $\|S\|-1$  clusterings. Now, Phase 3 performs a scan over the set of original artifacts and assigns each one of them to the cluster to which it is the closest. The complexity of this phase is  $O(\|S\|^2 qn)$ .

*Phase 4 (Determining the number of clusters):* This is the final phase. In this phase the differences among the clusters (produced in Phase 2) are tested against a threshold value to determine the appropriate number of clusters. If the difference between two clusters is less than the threshold then they are merged together to form a single one. In this way, the algorithm produces a refined set of clusters.

### 2.3.3 A Heuristic Approach to Solving the Software Clustering Problem

This approach proposed by Mitchell [3] is a heuristic one which creates random number of partitions and evaluates the “quality” of these partitions using a fitness function that is called Modularization Quality ( $MQ$ ) which has the property of redrawing cohesive clusters and penalizing excessive inter-cluster coupling. To find appropriate number of



clusters with appropriate software artifacts metaheuristic search algorithms are used. Given that the fitness of an individual partition can be measured with  $MQ$ , metaheuristic search algorithms are used in the clustering phase in an attempt to improve the  $MQ$  of the randomly generated partition. The name of the tool for implementing the heuristic approach is *BUNCH* which includes several hill climbing and a genetic algorithm also. The complexity of this algorithm is NP complete. The output created by *BUNCH* for a small compiler shown in Figure 2.4, is shown in Figure 2.5.

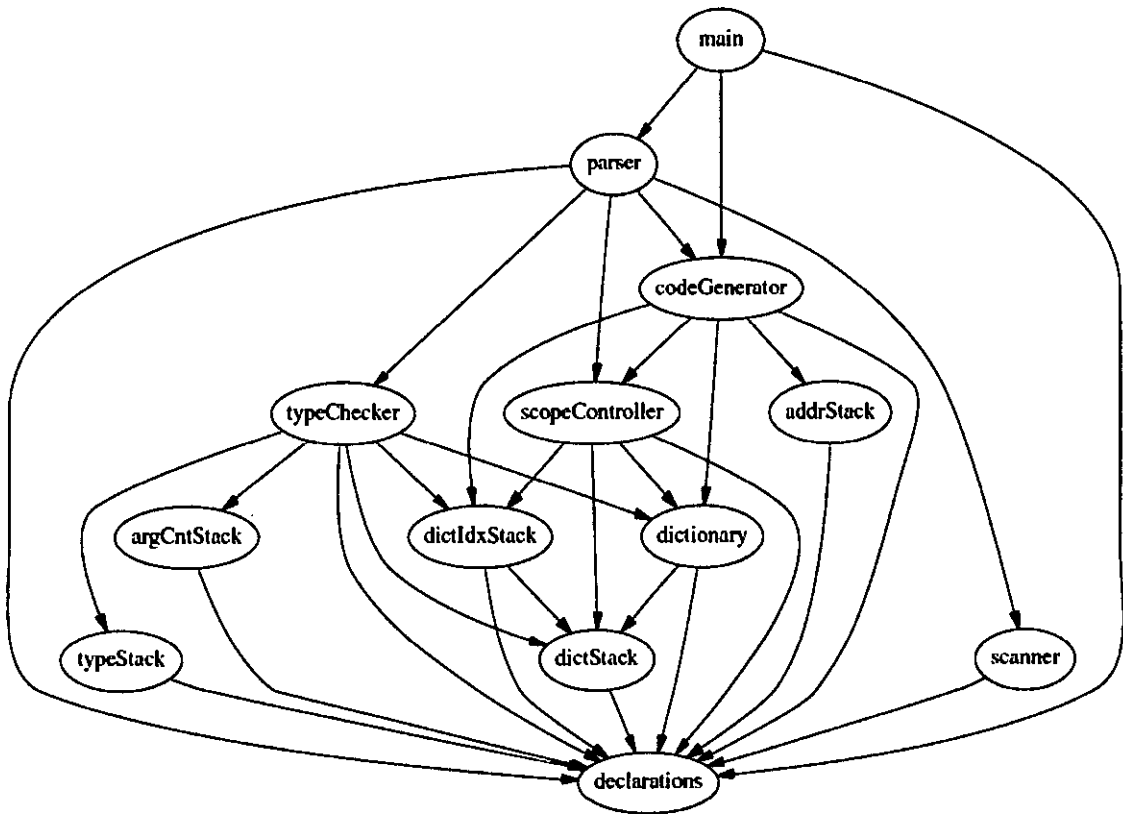


Fig. 2.4 A small compiler [3]

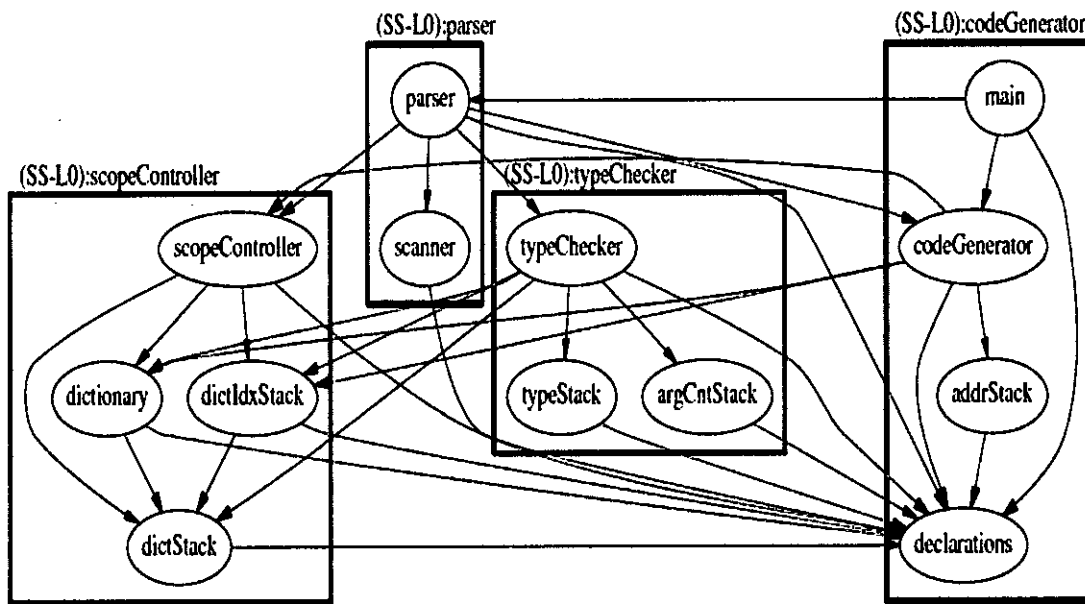


Fig. 2.5 Output from BUNCH for the small compiler [3]

### 2.3.4 Reverse Engineering Software Architecture using Rough Clusters

In this approach, Jahnke et. al. [4] argued that - while great diversity of algorithms on how to compute software clusters has been developed over time, they are not well adopted in industrial practice so far. The authors believe that this is mainly due to two main limitations, firstly, the lack of algorithms to represent approximate clusters, and secondly, the inability of clustering algorithms to use human expertise and domain knowledge about the legacy application.

For the solution to the first problem, the authors proposed to pin-point clusters based on the selection of so-called *seed artifacts* (*seeds* for short). Seeds are software artifacts that, according to a human expert, clearly belong to a particular cluster as defined in the domain architecture. The relationships of seeds to other software artifacts are then evaluated to build the basis for automatic clustering.

For the solution to the second problem, the authors proposed to adopt rough set theory to address this problem of representing ambiguity. Intuitively, a rough set  $\underline{S}$  uses two (traditional) sets  $S_{low}$  and  $S_{upp}$  to approximate another set  $S$ .  $S_{low}$  is called *lower approximation* and contains only those elements that are known to belong to  $S$ , while  $S_{upp}$  is called *upper approximation* and contains all elements that may belong to  $S$ . This model provides us with a simple, yet powerful way to address the ambiguity problem in

software clustering: if we know with certainty that a software artifact belongs to a given cluster, we add it to the lower approximation of this cluster. Otherwise, if we are uncertain about the membership of an artifact in a cluster, we can add it to its upper approximation. The overall approximation of rough clusters is shown in Figure 2.6.

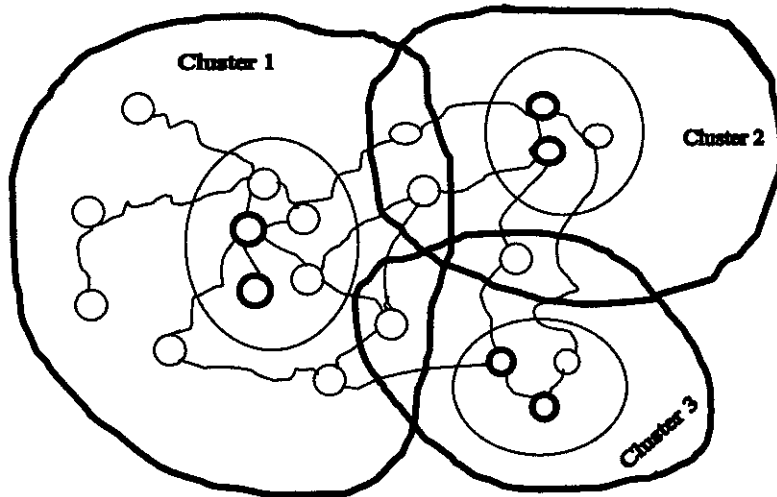


Fig. 2.6 Rough clusters

*Rough Set Theory Applied to Software Clustering:* Software clustering algorithms commonly use an abstract model about the interdependencies among software artifacts, which has been extracted from the system to be reverse engineered. We use an Artifact Dependency Graph (*ADG*) for this purpose, which can be formalized as a tuple  $ADG \leftarrow (A, R, K, t)$ , where

- $A$  is a finite set of software artifacts.
- $R$  finite multi set of dependencies among artifacts.
- $K$  is an alphabet of type labels.
- $t: A \cup R \rightarrow K$  is a labeling function, providing types for artifacts and dependencies (e.g., “class”, “variable”, “function”, “function call”, “uses” etc.).

It is the objective of clustering to find a (meaningful) partition of  $A$ . The authors believe that clusters only have meaning if we can identify at least one seed artifact in the legacy

system that belongs to it. So, this seed artifact would appear in the lower approximation of the cluster and every cluster will be strictly non-empty.

*Incremental Clustering Process:* The authors believe that user involvement is essential for the reverse engineering of meaningful software architectures from legacy code. Therefore, the clustering process suggested here has semi-automatic and incremental nature. It consists of two main phases, namely *Concept Assignment* and *Partition Refinement*. The first phase is a top-down analysis of the software system, while the second phase is primarily a bottom-up clustering.

The objective of Phase 1 is (*Concept Assignment*) to come up with a first-cut rough partition of the software system to be clustered based on user-defined conceptual model. The user defines a Business-Concept Model (*BCM*) describing the problem domain implemented in the legacy software system. The next step (*seed matching*) finds artifacts in the legacy software that can be associated with the concepts in the *BCM*. This step is done automatically based on simple string pattern matching. The following step (*impedance based clustering*) evaluates the dependencies between seed artifacts and other artifacts by using a weightage function in order to populate the rough clusters. In this approach, the user can parameterize the impedance-based clustering algorithm with two threshold values, namely the *lower threshold (LT)* and the *higher threshold (HT)* to populate lower approximation and upper approximation respectively. When the impedance-based clustering step is finished, the quality metrics are computed and the result is displayed to the user. After investigating the rough partition, the user can decide whether the match between the generated clusters and the *BCM* is satisfactory. It might take several iterations in Phase 1 (*Conceptual Assignment*) until the user is satisfied with the conceptual partitions (*generated clusters*) matched to the *BCM*. During these iterations, the user can modify seed assignments and change the *BCM* to attain a better match.

Phase 2 of the interactive clustering process (*Partition Refinement*) starts when the user is satisfied with the result of the conceptual match achieved in Phase 1. In this second phase, the user can investigate and resolve ambiguities by moving artifacts from the boundary regions to the lower approximation of clusters. Furthermore, the user can

resolve situations of incompleteness by creating new clusters and assign new seeds. The user can also customize the values for *HT* and *LT*, and the weights associated with the different dependency types. Like the first phase, the refinement phase is iterative and the user can re-evaluate the rough partition after each cycle of modifications. The process ends when the user is satisfied with the partition attained.

### 2.3.5 Other Prominent Clustering Approaches

In the early stage of software clustering, Hutchens et. al. [5] introduced the concept of data binding. A data binding classifies the similarity of two procedures based on the common variables in the static scope of the procedures. Anquetil et. al. [6] claimed that clustering based on naming convention often produce better results than using source code features. They presented several case studies that showed very promising results (high precision and high recall) using the file name features to cluster the software systems. Schwanke [7] provided a semi-automatic approach. His clustering heuristics were based on the principle of maximizing the cohesion of procedures placed in the same module and at the same time minimizing the coupling between procedures that reside in different modules. Slonim et. al. [8] presented *AIB* based on mutual information between two variables. Recently, Xiao et. al. [9] proposed software clustering based on dynamic dependencies.

## 2.4 Observations

In the previous section, we reviewed various prominent software clustering approaches. By examining these works, several interesting observations can be made –

- The majority of these clustering approaches use only the information of coupling and cohesion for similarity measurement.
- *ACDC* uses pattern matching for the generation of a skeleton by identifying the subsystems of the final decomposition. Depending on the pattern used by the subsystems, they are given appropriate names.
- *LIMBO* creates the summary artifacts from the original software artifacts by measuring the distance against a predefined threshold value. *LIMBO* also determines the final number of clusters by testing the inter-cluster distances against another predefined threshold value.

- *BUNCH* creates random number of partitions and evaluates the “quality” of these partitions using a fitness function that is called Modularization Quality (*MQ*). To find appropriate number of clusters with appropriate software artifacts metaheuristic search algorithms are used.
- *Rough Set Clustering* is a semi-interactive one which allows users to inject his/her valuable domain knowledge while clustering process is on and therefore the algorithm is iterative and incremental. Rough software architectures (number of partitions and their properties) can be refined by the users while clustering.

## 2.5 Research Challenges

Peculiarity of the software artifacts as a subject of clustering causes a lot of hindrances to the researchers. We describe some of the interesting research challenges below –

- *Improving performance*: Software clustering is never a “silver bullet”. There is always a scope of improving the result of software clustering. Incorporation of runtime information would make it more precise.
- *Control on cluster generation*: In the case of software clustering, there should be a doctrine (a set of guidelines) which can influence the clustering tendency in the process of cluster generation. Otherwise the resultant clusters may not look like the subsystems of a software system.
- *Improving visualization of the result*: Since visualization is one of the key tenants of program understanding, improved visualization services would have high impact. There can be many levels of views based on appropriate levels of granularity. Also, the software system can have more than one valid view.
- *Other research opportunities*: Most of the clustering approaches are computationally intensive. We should find ways to make the software clustering approaches faster so that they can be applied to the extra large software systems to be dealt in future.

## 2.6 Chapter Summary

In this chapter we have presented how clustering can be used in the domain of software systems to discover the subsystems within them. Then, we have presented some prominent approaches of software clustering and we had some observations on those approaches. Finally we have explored some research challenges in the arena of software clustering.

## Chapter 3

### Software Clustering Using the Knowledgebase

As evidenced in the previous chapter, software clustering problem has attracted the attention of many researchers. In this chapter, we figured out some serious problems of those approaches presented earlier. As a consequence, we introduce a novel approach of software clustering based on Knowledgebase. Here we identify the need to utilize the power of Knowledgebase as the solution to many problems associated with software clustering. A detailed description along with pseudo code, flow chart and block diagram of our approach is presented too. The chapter concludes with comparative characteristic observations among the prominent approaches of software clustering.

#### 3.1 Problems with Existing Software Clustering Approaches

Appropriate similarity measurement is the most important issue in any clustering problem. In software clustering, similarity measurement is a challenge as wide range of diversity in nature is observed in the software systems. That is why there is no generalized similarity measurement technique applicable to every software system.

##### 3.1.1 Existing Similarity Measurement Techniques for Software Systems

Similarity measures like association coefficients, distance measures and correlation coefficients are not flexible enough to be used in software clustering. However, it has been observed that Jaccard coefficient [11] gives better result compared to other typical similarity measures for software clustering. Recently Maqbool et. al. [10] showed that incorporation of weighted scheme through Ellenberg measure in similarity measurement outperforms Jaccard coefficient and guarantees accumulation of more appropriate software artifacts in the process of clustering. Here we give a simple example from a Library Management System.

Suppose, in the Library Management System there are four entities like: Book, Rent, Book\_Find and Rent\_Find. The feature vectors used by the four entities are presented in Table 3.1 where presence of any feature is represented as 1 and absence as 0.

The features indicated in the feature vector are:



1. Use of the same Global Variables named
  - a. Name
  - b. Emp
  - c. Rent
  - d. Match
2. Function Call named
  - a. Book\_Find
  - b. Rent\_Find

*Table 3.1 Entity with their associated features*

| Entity    | Feature Vector       |
|-----------|----------------------|
| Book      | { (0 1 0 1), (0 1) } |
| Rent      | { (0 1 0 1), (1 0) } |
| Book_Find | { (1 0 1 0), (0 1) } |
| Rent_Find | { (1 0 1 0), (1 0) } |

The corresponding similarity table using the Jaccard coefficient is tabulated in Table 3.2.

*Table 3.2 Similarity table using Jaccard coefficient*

|           | Book | Rent | Book_Find | Rent_Find |
|-----------|------|------|-----------|-----------|
| Book      | –    | 1    | 1/4       | 0         |
| Rent      | –    | –    | 0         | 1/4       |
| Book_Find | –    | –    | –         | 1         |
| Rent_Find | –    | –    | –         | –         |

Here, in the first step of generating clusters, Book and Rent will be merged to form a new cluster. In the second step, Book\_Find and Rent\_Find will be merged to form another new cluster. But normally, Book and Rent should be placed into different clusters as they are totally separate entities. Moreover, Book\_Find should belong to the same cluster of Book and Rent\_Find should belong to the same cluster of Rent.

Now, if we consider a run-time information we find that the two functions Book\_Find and Rent\_Find are called by the functions Book and Rent respectively multiple times

(say, 5 times) and it is expected that such called functions and callee entities should be in the same clusters. Thus to incorporate this obvious nature of clustering we can assign weightages for multiple function calling in the expression of similarity measurement. In this scenario the feature vectors of the entities can be presented as shown in Table 3.3.

*Table 3.3 Entity with their associated features*

| <b>Entity</b> | <b>Feature Vector</b> |
|---------------|-----------------------|
| Book          | { (0 1 0 1), (0 5) }  |
| Rent          | { (0 1 0 1), (5 0) }  |
| Book_Find     | { (1 0 1 0), (0 5) }  |
| Rent_Find     | { (1 0 1 0), (5 0) }  |

Now, by applying Ellenberg measure on the feature vector presented in Table 3.3 we get the following similarity table as shown in Table 3.4.

*Table 3.4 Similarity table using weighted scheme*

|                  | <b>Book</b> | <b>Rent</b> | <b>Book_Find</b> | <b>Rent_Find</b> |
|------------------|-------------|-------------|------------------|------------------|
| <b>Book</b>      | –           | 1/6         | 5/9              | 0                |
| <b>Rent</b>      | –           | –           | 0                | 5/9              |
| <b>Book_Find</b> | –           | –           | –                | 1/6              |
| <b>Rent_Find</b> | –           | –           | –                | –                |

This time, in the first step, Book and Book\_Find will be merged to form a new cluster and in the second step, Rent and Rent\_Find will be merged to form another new cluster. Certainly, these resultant clusters are more appropriate compared to the previous result. But this run-time information of function calling is a real-time information where the number of calls of a particular function can be different based on different tasks performed by the user. Also, the run-time information is not easily accessible and can not be obtained from the available source code. So, assigning weightage based on run-time function call is not a realistic option for software clustering.

As we see, all the similarity measurement processes used in different software clustering approaches like Ellenberg measure (for weighted scheme), Orphan Adoption [1] (in

*ACDC*) , Summary Artifacts Creation [2] (in *LIMBO*, where measuring the distance is vital) or Modularization Quality Calculation [3] (in the Heuristic Approach) do not consider the diversity and uniqueness among the software systems. As a result, the common drawbacks of these approaches are:

- They propose a generalized similarity measurement criterion for all the software systems. But in reality, even one software system may consider different similarity measurement criteria in different scenarios or environments.
- They can not incorporate dynamic weightages on the selected similarity measurement criteria. But in reality, one similarity measurement criteria may have different importance for different software systems.

Thus, the similarity measurement process of software clustering should be a dynamic one to cope up with the diversity in software systems. It is worth mentioning that a particular weightage of a similarity measurement criterion is denoted by the strength of the feature to contribute in the similarity among several entities.

### 3.1.2 Our Solution to this Problem: Software Clustering using the Knowledgebase

Almost every software system can be grouped into some Generic types such as Database Management Systems (*DBMS*), Operating Systems (*OS*), Compilers, Network utilities, Games etc. Most of the software systems under the same Generic type share many common characteristics. They can have many similar approaches, functionalities and features. Also they can have similar inputs and/or outputs. So, it is very likely that they will have similar types of considerations for similarity measurement. For example, a *DBMS* can consider a database table name or prefix of any software artifact name as the most important in similarity measurement criteria, a compiler can consider the global variable usage / computational aspects of functions or an *OS* can give emphasis on high cohesion and low coupling. So, Generic type of any software system can provide strong clues for similarity measurement by providing guidelines on similarity measurement criteria and weightage for the respective similarity measurement criteria.

The Generic type of any software system not only provides direction for similarity measurement but also provides valuable information about the formation (structural

arrangement) of that software system. As a result, indispensable subsystems of a software system can also be comprehended according to its Generic type.

Now, if a software clustering approach is able to utilize the information about the Generic type of the software system to be clustered, the overall quality of the clustering result will be improved. So, the use of Knowledgebase comes forward as a consequence.

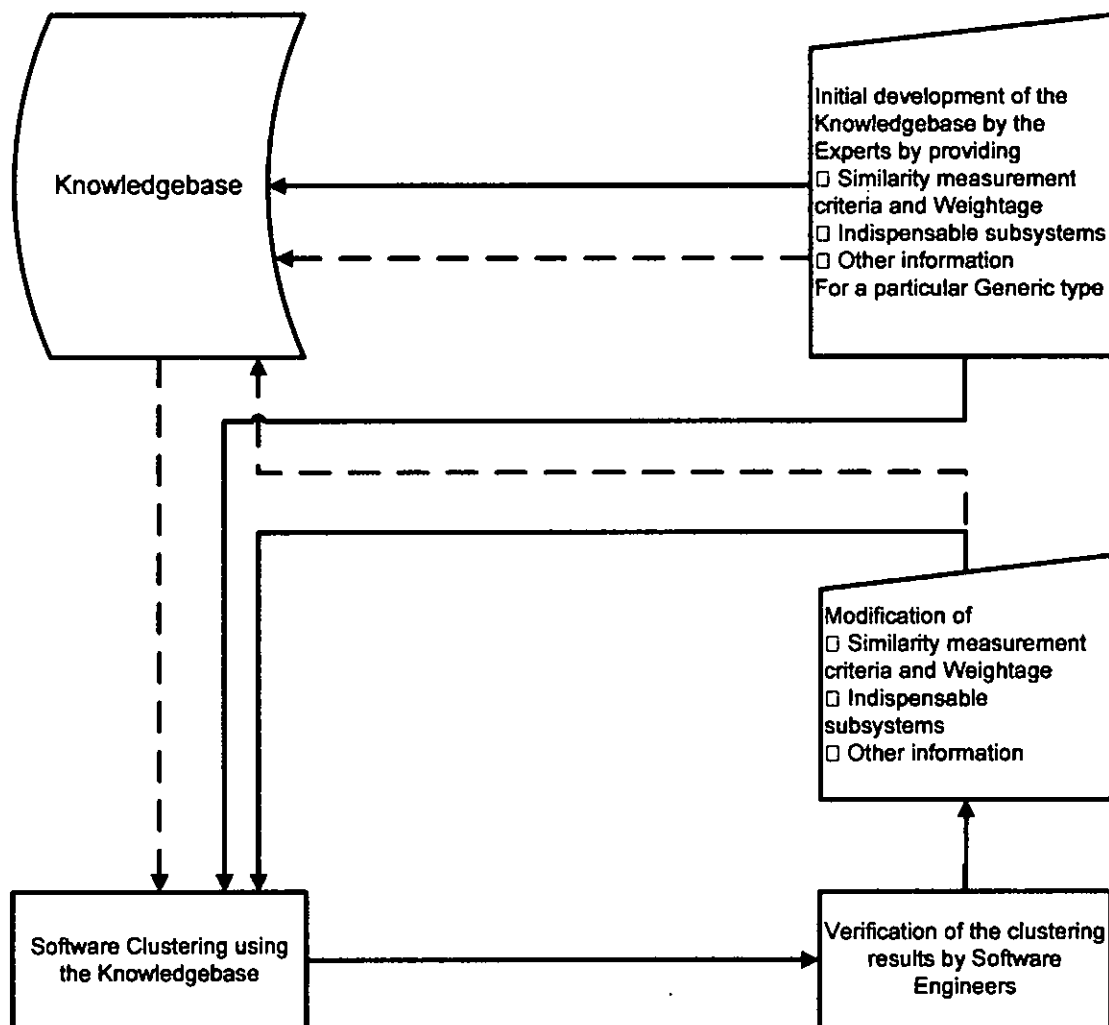
## 3.2 The Knowledgebase

Knowledgebase is a common term used in the discipline of artificial intelligence defined as collection of rules and facts relevant to the real world [13]. In the context of Software clustering, the Knowledgebase will act as a repository of information about the internal structure of the Generic types of software systems. The Knowledgebase will contain information about how the software artifacts (under the same Generic type) are inter-related (the similarity measurement criteria and their respective weightages) and how they differ from each other. The Knowledgebase may also contain information like indispensable subsystems, program unit complexity, cyclomatic complexities etc. of the Generic types of software systems.

### 3.2.1 Knowledgebase Development

Development of any type of Knowledgebase is a continuous process. The Knowledgebase for software clustering is not an exception. However, the major challenge here is to identify the rules and facts associated with the Generic types of software systems and incorporate them into the Knowledgebase. But in our proposed technique, rules and facts (i.e. similarity measurement criteria and their respective weightages, indispensable subsystems etc.) relating to the Generic types will be observed by the experts and software engineers will apply them in software clustering. As a result, the software engineers will be able to verify the effectiveness of those rules and facts used in the clustering process and if necessary modify them to generate more appropriate clustering.

So, the Knowledgebase for software clustering is an adaptive technique to improve the generation of more appropriate clustering from a continuous learning process by the experience of clustering. Figure 3.1 shows the Knowledgebase development process –



*Fig. 3.1 Knowledgebase Development*

### 3.2.2 Knowledgebase Representation

The information of the Knowledgebase can be stored in a relational database [14]. The Knowledgebase representation (using the relational database) can be further enhanced with inference mechanism like property inheritance, in which elements of specific classes inherit attributes and values from more general classes in which they are included. In this research, the information about similarity measurement and subsystem identification has been incorporated into the relational database. Here we present some of the facts of the Generic types:

Table 3.5 Facts associated with the Generic Types

Fact 1: (For DBMS)

| Similarity Measurement Criteria              | Weightage |
|--|-----------|
| Function Call (A function calls another one) | 25.00     |
| Same Global Variable Use                     | 2.00      |
| Two functions calling the same function      | 15.00     |
| Shares the Same Prefix (i.e. Employee*)      | 100.00    |
| Shares the Same Table                        | 1000.00   |
| Same Global Structure/Class Use              | 5.00      |

Fact 2: (For OS)

| Similarity Measurement Criteria              | Weightage |
|--|-----------|
| Function Call (A function calls another one) | 100.00    |
| Same Global Variable Use                     | 5.00      |
| Two functions calling the same function      | 75.00     |
| Shares the Same Prefix (i.e. init*)          | 25.00     |
| Same Global Structure/Class Use              | 5.00      |

Fact 3: (For Compilers)

| Similarity Measurement Criteria              | Weightage |
|--|-----------|
| Function Call (A function calls another one) | 15.00     |
| Same Global Variable Use                     | 75.00     |
| Two functions calling the same function      | 10.00     |
| Shares the Same Prefix (i.e. parse*)         | 50.00     |
| Same Global Structure/Class Use              | 100.00    |

...  
 ...  
 ...  
 ...

Fact n: (For Games)

| Similarity Measurement Criteria              | Weightage |
|--|-----------|
| Function Call (A function calls another one) | 15.00     |
| Same Global Variable Use                     | 125.00    |
| Two functions calling the same function      | 5.00      |
| Shares the Same Prefix (i.e. render*)        | 75.00     |
| Same Global Structure/Class Use              | 100.00    |

An example showing representation of similarity measurement criteria and their respective weightages in a relational database is given as follows:

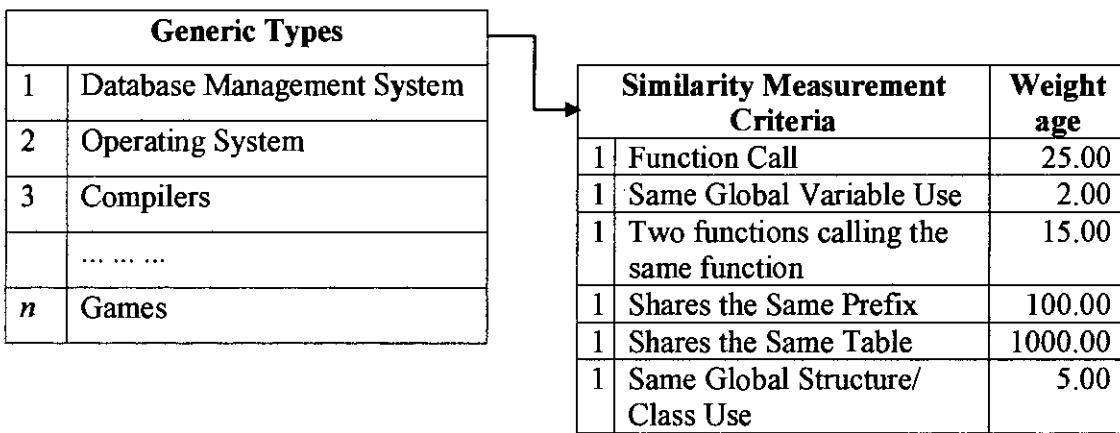


Fig. 3.2 Simple relational Knowledgebase for similarity measurement

### 3.3 Similarity Measurement using the Knowledgebase

Now the process of similarity measurement comes to the point. At first, the proper Generic type of the software system to be clustered is determined. Then the appropriate similarity measurement criteria and their respective weightages for that particular Generic type are obtained from the Knowledgebase. The Strength of Similarity ( $\sigma$ ) between two software artifacts can be computed using the following expression:

$$\sigma(A, B) = \sum_{i=1}^m w_i A_i B_i$$

Here,  $A$  and  $B$  are two software artifacts.

Index  $i$  indicates a similarity measurement criteria of the Generic type.

$m$  is the number of similarity measurement criteria considered for similarity measurement of the corresponding Generic type of the software system.

$w$  is the weightage vector and  $w_i$  indicates the weightage for the  $i$ -th similarity measurement criteria.

$A_i$  and  $B_i$  indicates the absence and presence of a feature in entity  $A$  and  $B$ . That is,  $A_i, B_i = \{0, 1\}$ .

### 3.3.1 Example of Similarity Measurement Computation –

Here we apply similarity measurement criteria with their respective weightages of *DBMS* from Table 3.5 to a Library Management System.

Now, we consider two functions *Employee\_add* and *Employee\_edit* of the Library Management System to compute the Strength of Similarity between them.

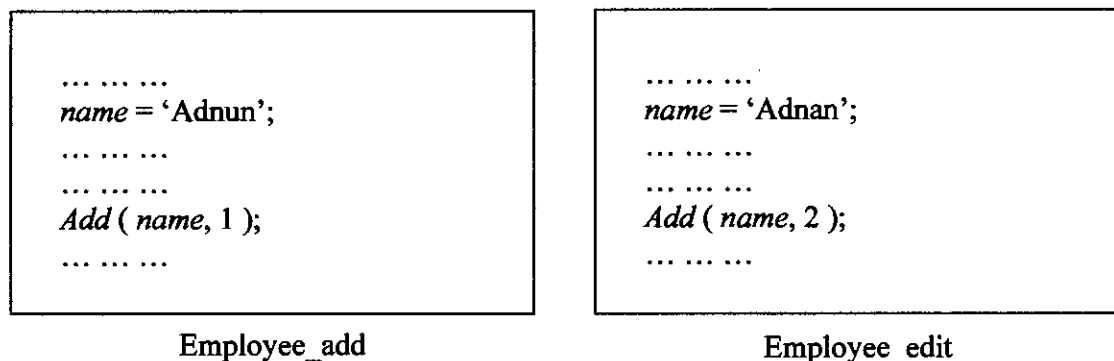


Fig. 3.3 A typical example of two functions

According to the similarity measurement criteria and their respective weightages of Table 3.5, the components of the Strength of Similarity between these two functions will be:

- For using the same global variable (*name*): 2
- For calling the same function (*Add*): 15
- For sharing the same prefix (*Employee\**): 100

So, the total Strength of Similarity will be: 2+15+100=117.



After the computation of the Strength of Similarity among all pairs of software artifacts of a given software system, we get the desired Similarity Matrix. Here we present an example of Similarity Matrix to get a visual overview:

*Table 3.6 Example of a part of the Similarity Matrix*

|                      | <b>Book_Find</b> | <b>Rent_Find</b> | <b>Employee_Add</b> | <b>Employee_Edit</b> |
|----------------------|------------------|------------------|---------------------|----------------------|
| <b>Book_Find</b>     | –                | 17               | 2                   | 2                    |
| <b>Rent_Find</b>     | –                | –                | 2                   | 2                    |
| <b>Employee_Add</b>  | –                | –                | –                   | 117                  |
| <b>Employee_Edit</b> | –                | –                | –                   | –                    |

### 3.4 Subsystem Identification using the Knowledgebase

There may have some indispensable subsystems associated with the Generic type of any software system. Even the software system to be clustered may have some known obligatory subsystems with some known properties. For example, a Library Management System can have subsystems like Book, Member etc. Here we label these indispensable subsystems as *soul* clusters. These obligatory subsystems for any software system or of its Generic type can be stored into the Knowledgebase to ensure their presence in the clustering result for the correctness of the overall clustering. As the soul clusters (if any) are distinguishing to their associated software systems, the presence of them in the clustering result makes the result more acceptable to the software engineers.

There can be many types of pre-conditions based on which a software artifact can be accumulated into a soul cluster. Some of them are listed as follows:

- Possessing the same predefined prefix.
- Calling the same predefined function or routine.
- Reading from the same type of files.
- Writing to the same type of files.
- Functions with similar cyclomatic complexity.

### 3.5 Software Clustering Using the Knowledgebase

The proposed clustering approach requires some preprocessing before applying the regular algorithm for clustering. The basic methodologies of both the steps are described as follows:

#### *Preprocessing of Software Artifacts:*

The following operations are performed in this step:

- All the software artifacts are compared with the soul clusters defined in the Knowledgebase. The software artifacts that satisfy the pre-conditions to be a part of the soul clusters are no more considered as separate entities and they are merged into the soul clusters.
- A Similarity Matrix is determined with the remaining software artifacts which do not match with any of the soul clusters.
- Two clusters with the highest similarity measurement from the Similarity Matrix will be clustered to form a *premature* cluster. This process of forming premature cluster will be repeated on the remaining software entities of the Similarity Matrix.
- It is worth mentioning that there will be a single software artifact remaining without forming premature cluster if there is odd number of software artifacts in the Similarity Matrix. We define that software artifact as a premature cluster as well. The main objective of finding premature cluster is to reduce the search space substantially.

Thus the outcome of the preprocessing step is a collection of soul clusters and premature clusters. We define each of these clusters as *candidate* cluster.

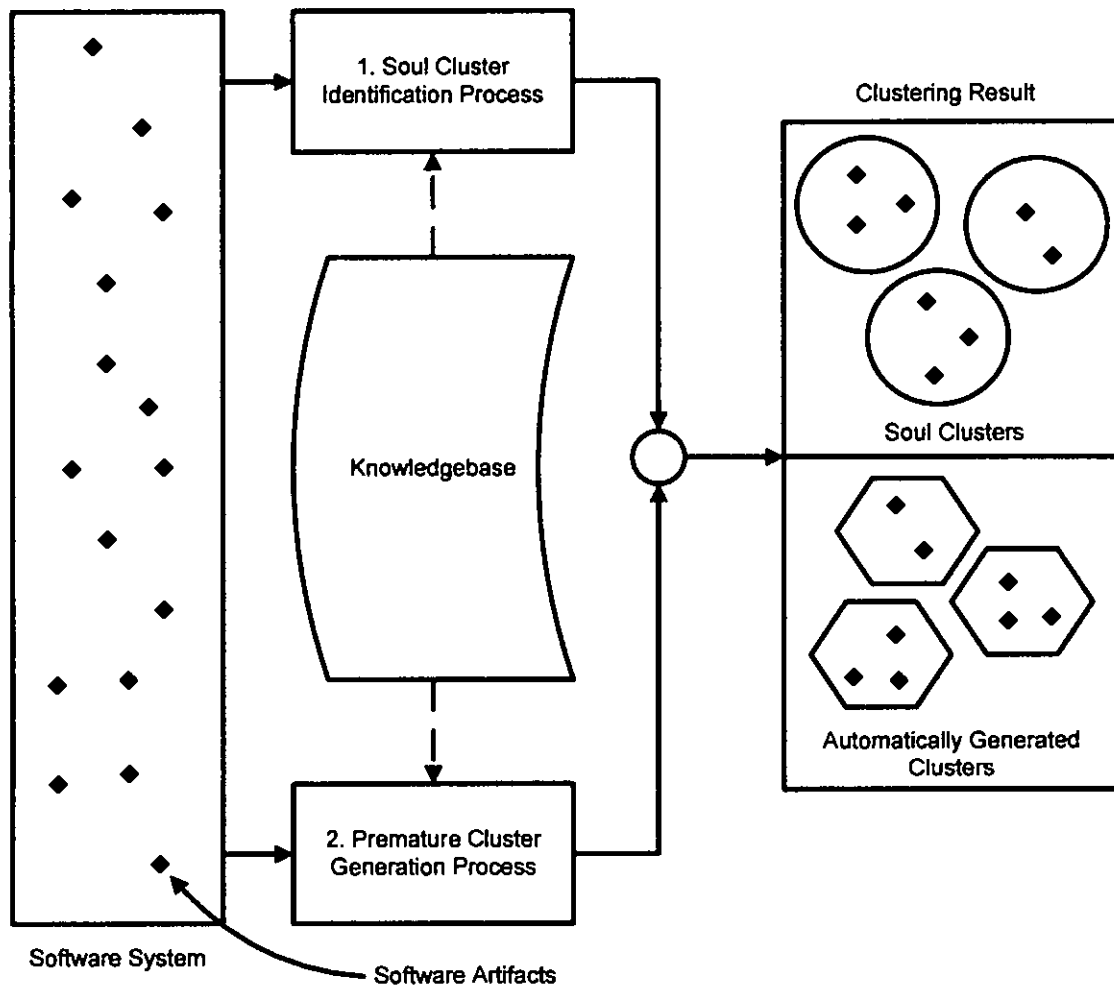
#### *Clustering of Candidate Clusters:*

The operations performed in this step are straightforward. In each iteration of this step the following operations are performed:

- A new Similarity Matrix is determined with the candidate clusters.
- Two candidate clusters with maximum strength of similarity is merged into a new candidate cluster.

The iteration stops when the number candidate cluster reduces to a threshold value, defined in the Knowledgebase. This threshold can be a user input of the clustering system from the software engineer.

The overall architecture of the clustering process can be presented in the block diagram shown in Figure 3.4.



*Fig. 3.4 Architecture of the proposed clustering approach*

The flow chart of the clustering system is shown in Figure 3.5.

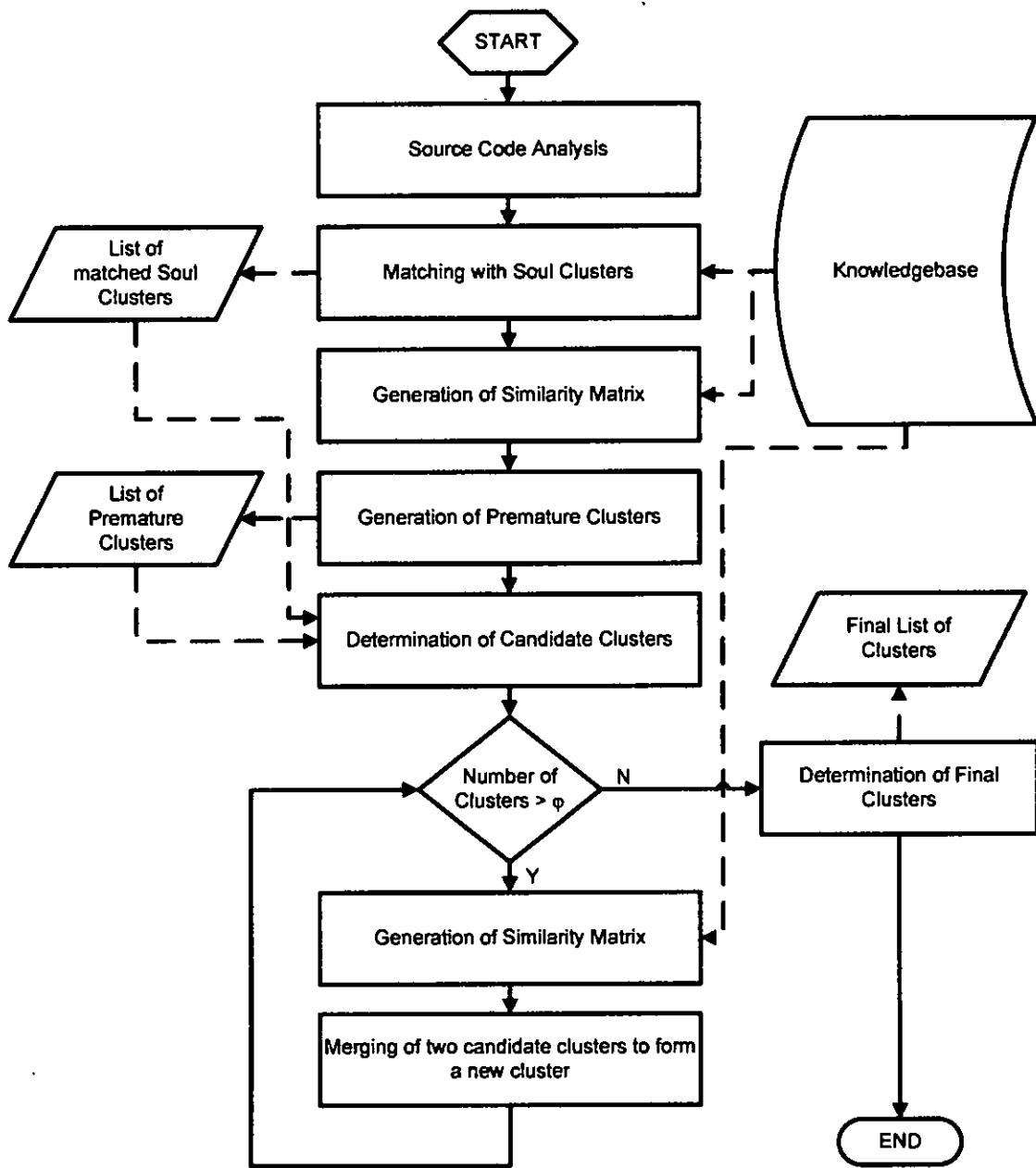


Fig. 3.5 Flow Chart of the proposed clustering process

### 3.6 Pseudo Codes

Here we present the pseudo codes for Similarity Matrix generation, premature cluster generation and final clustering from the candidate clusters:

*Procedure Generate\_Similarity\_Matrix (A)*

*begin procedure*

// A: The list representing the feature of vectors of entities  $A_1, A_2, \dots, A_n$ .  $A_i$  is the  $i$ -th entity (software artifact) and  $A_{ik}$  is the  $k$ -th feature of the  $i$ -th entity.

for  $i \leftarrow 1$  to  $n$  do //  $n$  is total number of software artifacts

for  $j \leftarrow i + 1$  to  $n$  do

$\sigma(i, j) \leftarrow 0$  //  $\sigma(i, j)$  is the strength of similarity between  $i$ -th and  $j$ -th entity

for  $k \leftarrow 1$  to  $m$  do

$\sigma(i, j) \leftarrow \sigma(i, j) + w_k A_{ik} A_{jk}$

end for

end for

end for

return  $\sigma$  //  $\sigma$  is the matrix of strength of similarities

*end procedure*

*Procedure Generate\_Candidate\_Cluster (A)*

*begin procedure*

// A: The list representing the feature of vectors of entities  $A_1, A_2, \dots, A_n$ .

$soul\_clusters = match\_soul\_cluster(A)$  // Matching the soul clusters in  
//the Knowledgebase

$individual\_clusters = A - soul\_clusters$

// Taking out soul cluster for generation of the next

// similarity measurement calculation

$\sigma \leftarrow Generate\_Similarity\_Matrix(individual\_clusters)$

for  $i \leftarrow 1$  to  $size\_of(individual\_clusters)$  do  $is\_clustered_i = false$

// Marking that no entity has been selected as premature cluster

while  $size\_of(individual\_clusters) > 1$  do // Finding a premature cluster in each step

$(i, j) = Find\_Max\_Strength(individual\_clusters, is\_clustered, \sigma)$

// Finding a couple of entities with maximum

// strength of similarity

```

C ← merge ( Ai, Aj ) //Merging two entities with maximum strength of similarity
is_clusteredi ← true //Discarding the merged entity from the list
is_clusteredj ← true //of consideration for premature cluster
individual_clusters ← individual_clusters - Ai - Aj
premature_clusters ← premature_clusters + C
end while
if size_of(individual_clusters) = 1 then
    premature_clusters = premature_clusters + A1 //Number of individual entities is
end if //odd
return (premature_clusters + soul_clusters)
end procedure

```

*Procedure Find\_Max\_Strength* (*individual\_clusters*, *is\_clustered*,  $\sigma$ )

*begin procedure*

*max\_strength* = 0

for *i* ← 1 to size\_of(*individual\_clusters*) do //Searching in the Similarity Matrix

for *j* ← *i* + 1 to size\_of(*individual\_clusters*) do

if *is\_clustered<sub>i</sub>* = false and *is\_clustered<sub>j</sub>* = false then

//Already clustered entities will not be considered

if  $\sigma_{ij} > \text{max\_strength}$  then //This strength is higher than the  
//previous one

*max\_strength* =  $\sigma_{ij}$

*index\_1* = *i*

*index\_2* = *j*

end if

end if

end for

end for

return (*index\_1*, *index\_2*) // Returning the indices with the highest strength

end procedure

```

Procedure Do_Final_Clustering (candidate_clusters,  $\phi$ )
begin procedure
//  $\phi$  is the threshold value to determine the final number of clusters
while size_of (candidate_clusters) >  $\phi$  do
     $\sigma$  = Generate_Similarity_Matrix (candidate_clusters)
    for  $i \leftarrow 1$  to size_of (individual_clusters) do  $is\_clustered_i = false$ 
    //Intializing is_clustered indicating that none of the entities will be discarded
     $(i, j)$  = Find_Max_Strength (candidate_clusters, is_clustered,  $\sigma$ )
     $C \leftarrow merge (A_i, A_j)$  //Merging two entities with maximum
    //strength of similarity
    candidate_clusters  $\leftarrow candidate\_clusters - A_i - A_j$ 
    candidate_clusters  $\leftarrow candidate\_clusters + C$ 
end while

```

### 3.7 Comparison of the Proposed New Clustering Method over the Existing Methods

The following points depict why the proposed method is exceptional from the existing ones:

- The proposed approach can combine both the soul clusters and automatically generated clusters in the same result. No other approaches proposed so far, can do this. Rough set clustering approach can detect only user-defined subsystems configured in the *BCM*. Users might not define every subsystem of a huge software system. As a result, many subsystems can be absent in the clustering result which can lead to wrong clustering as a whole.
- All the clustering approaches proposed so far, did not consider these soul clusters; rather they relied on the clustering generated by the process which could lead to the absence of many of these types of soul clusters. As a result, due to the absence of many crucial subsystems the whole clustering result will be meaningless to the software engineers.
- A soul cluster defined in the Knowledgebase may not be a part of the software system to be clustered. Our approach ignores those soul clusters which do not

match any of the software entities. Thus our newly proposed system is not totally dependent on the Knowledgebase. It incorporates the benefits of the Knowledgebase through the software artifact criteria extracted from the code of the software system.

- In any software clustering approach, the threshold value which determines the number of clusters in the final result is very important. The total skeleton of the resultant clustering depends on this value. The Heuristic Approach (*BUNCH*) randomly generates the threshold value. If the random generation is affected by local minima, the threshold value will not be close to an acceptable one. So, important subsystems can be divided or ignored in the clustering result. As the threshold value is generated randomly, the skeleton may differ in the second run. Different results from different runs are not realistic as it can create confusion to the software engineers.
- Finally, there no room for improvement for the clustering approaches proposed so far. But our proposed approach is an exceptional in this regard. Our clustering approach learns to cluster more efficiently from the experience of clustering and day by day it improves its capability to discover the underlying subsystems of the software systems.

### 3.8 Complexity Analysis

As we know already, there are two main steps in our clustering approach. They are: preprocessing for software artifacts and clustering of candidate clusters. In both the steps Similarity Matrix calculation contributes to the complexity analysis substantially. That is why we present the complexity of Similarity Matrix at first.

#### *Complexity of Similarity Matrix Calculation:*

For  $n$  artifacts there will be  ${}^n C_2$  entries for  ${}^n C_2$  pairs of entities in the Similarity Matrix. For the calculation of the strength of similarity for a pair of entity it requires  $m$  floating point operations. Here  $m$  is the number of features in the feature vector of the entities. Thus the total complexity of Similarity Matrix calculation will be  $mn(n-1)/2$ .



*Complexity of preprocessing for software artifacts:*

The number of floating point operations in this step are as follows:

- $mn_s p$  comparisons where we have  $n_s$  soul clusters in the Knowledgebase and there are  $p$  criteria for matching a soul cluster against  $n$  entities.
- $m(n_r)(n_r - 1)/2$  additions to calculate Similarity Matrix where we have  $n_r$  remaining entities which do not belong to the soul clusters.
- $\sum_{i=1}^{n_r/2} \frac{(n_r - 2i + 2)(n_r - 2i + 1)}{2}$  searching operations for finding  $n_r/2$  premature clusters from  $n_r$  remaining entities. It is worth mentioning that we need to do  $\frac{(n_r - 2i + 2)(n_r - 2i + 1)}{2}$  searching operations to find  $i$ -th premature cluster in the  $i$ -th iteration as shown in the *while loop* of Procedure *Generate\_Candidate\_Cluster*.

For the simplification the complexity calculation we can assume that  $n_s \approx n/4$  and  $n_r \approx n/2$  then the number of floating point operations in this step is expressed as:

$$C_{Preprocessing} = \frac{n^2 p}{4} + \frac{mn(n-1)}{4} + \sum_{i=1}^{n/4} \frac{(n-4i+4)(n-4i+2)}{8} = O(n^3)$$

*Complexity of clustering of candidate clusters:*

Let there be  $n_c$  candidate clusters in the beginning of this step. In each iteration of the *while loop* in procedure *Do\_Final\_Clustering* we merge two clusters into a single cluster.

In  $i$ -th iteration of this loop the number of floating point operations are as follows:

- $\frac{m(n_c - i + 1)(n_c - i)}{2}$  additions to calculate Similarity Matrix where we have  $n_c - i + 1$  candidate clusters.
- $\frac{(n_c - i + 1)(n_c - i)}{2}$  searching operations to find maximum strength of similarity in a Similarity Matrix with  $n_c - i + 1$  candidate clusters.

This clustering will be done until the number of cluster reduces to  $\varphi$ . For the simplification of the complexity calculation we can assume that  $n_c \approx n/2$  then the number of floating point operations in this step is expressed as:

$$C_{Final\_Clustering} = \sum_{i=1}^{n/2-\varphi-1} \left\{ \frac{m(n-2i+2)(n-2i)}{8} + \frac{(n-2i+2)(n-2i)}{8} \right\} = O(mn^3)$$

Thus the complexity of the total clustering process is expressed as:

$$\begin{aligned} C_{Clustering} &= C_{Preprocessing} + C_{Final\_Clustering} \\ &= \frac{n^2 p}{4} + \frac{mn(n-1)}{4} + \sum_{i=1}^{n/4} \frac{(n-4i+4)(n-4i+2)}{8} + \sum_{i=1}^{n/2-\varphi-1} \left\{ \frac{m(n-2i+2)(n-2i)}{8} + \frac{(n-2i+2)(n-2i)}{8} \right\} \\ &= O(mn^3) \end{aligned}$$

### 3.9 Chapter Summary

In this chapter we have presented our clustering approach which can exploit the Knowledgebase for the purpose of generating meaningful and legitimate subsystems. We also presented the problems that the existing approaches face on the way of software clustering and came up with a novel solution to those problems – “Introduction of the Knowledgebase in the arena of Software Clustering”. We also discussed the computational complexity of the main parts of our clustering approach. Also, this chapter includes block diagram, pseudo code and flow chart and of our approach. The next chapter describes the clustering capability of *BUET Cluster 1.0* (implementation of our approach presented in this chapter), *BUNCH* and *ACDC*.

# Chapter 4

## Implementation of Clustering System

In Chapter 3 we described our clustering approach, which generates subsystems using the Knowledgebase. We have a tool named *BUET Cluster 1.0* to demonstrate our clustering approach. This chapter primarily describes the structural design of *BUET Cluster 1.0* in detail along with APIs and other tools used in the processes of clustering. This chapter also describes another two well-known clustering tool *BUNCH* and *ACDC* shortly.

### 4.1 Key Modules in *BUET Cluster 1.0*

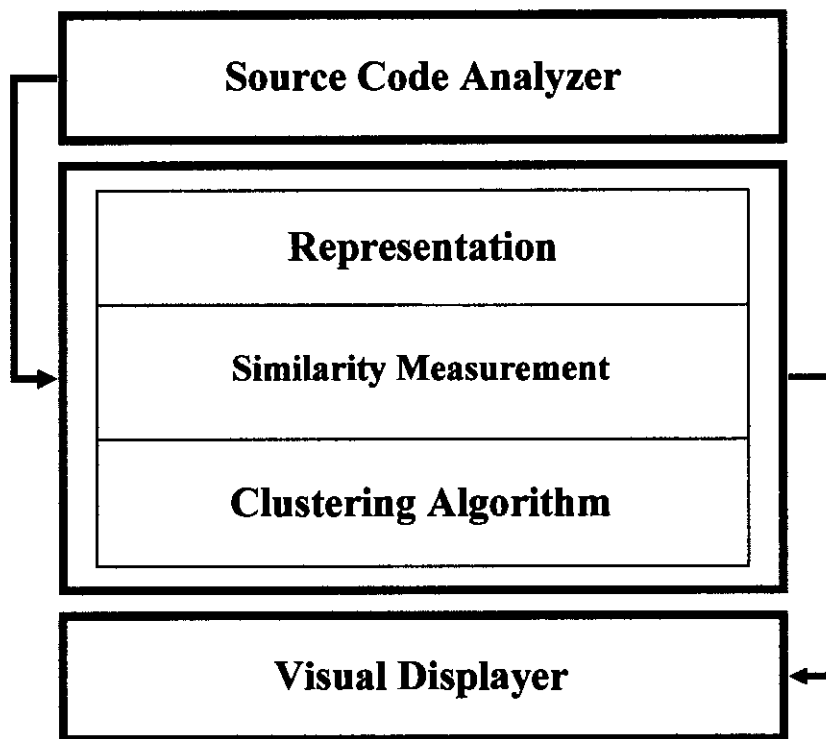
Almost every software clustering tool consists of some key modules where each module performs quite different tasks from others. These are defined briefly as follows:

1. *Source Code Analyzer*: This module involves source code analysis for creating a language independent representation called Module Dependency Graph [3] (*MDG*) based on the relations embedded in the source code. Source code analysis is done with the help of a specialized source code analysis tools like – *Understand 2.0* [15] of Scientific Toolworks, Inc. The result of the source code analysis is an intermediate representation of the source code. Next, this intermediate representation is parsed and an in-memory representation is generated which we distinguish as the *MDG*. The *MDG* can be used to view the whole un-clustered software system.
2. *Cluster Generator*: This is the most important module of any clustering approach. In this module, the *MDG* gets partitioned in such a way that those partitions represent subsystems of the software systems. To do this, some important tasks are carried out by this module:
  - i) Firstly, a modified representation of the source code is created from *MDG* which actually serves as the input to the clustering algorithm. This modified representation may vary as the format of the input may differ from one algorithm to another.
  - ii) Secondly, the clustering result is created from the modified representation by the clustering algorithm. This is the most

distinguishing part of any software clustering tool. That is – if you supply the modified representation of the source code acceptable to the clustering algorithm then you will be supplied with the clustering result in such a way which can be viewed with the available visualization tool.

3. *Visual Display*: This is the final module that involves visualization of the clustering results. Graph visualization tool such as *Graphviz-win* v2.16 [16] can be used to view the clustering results.

All these major modules with their sub-modules are shown in Figure 4.1.



*Fig. 4.1 Modules in clustering approaches*

Now, we will give a detailed description on how these steps are implemented.

## 4.2 Source Code Analysis and Creation of *MDG*

*BUET Cluster 1.0*, *BUNCH* and all other software clustering tools assume that the source code of a software system must be presented in such a way that adheres to a standard

format. For this, a suitable 3<sup>rd</sup> party source code analysis tool such as – *Understand 2.0* has been used.

#### 4.2.1 Source Code Analysis using *Understand 2.0*

*Understand 2.0* is a cross-platform, multi-language and maintenance oriented Interactive Development Environment (*IDE*). It is designed to help maintain and understand large amounts of legacy or newly created source code. The source code analysis by *Understand 2.0* may include Ada, C++, C#, FORTRAN, Java, JOVIAL and/or Delphi/Pascal. *Understand 2.0* creates an intermediate representation of the relations and structures contained within the source code of the software system.

On the way of creating the intermediate representation using *Understand 2.0*, the following steps are performed:

Step 1: *Understand 2.0* can be run directly from the menu shown in Figure 4.2 of our implemented project. The path is: **Code Analysis → Understand 2.0**.

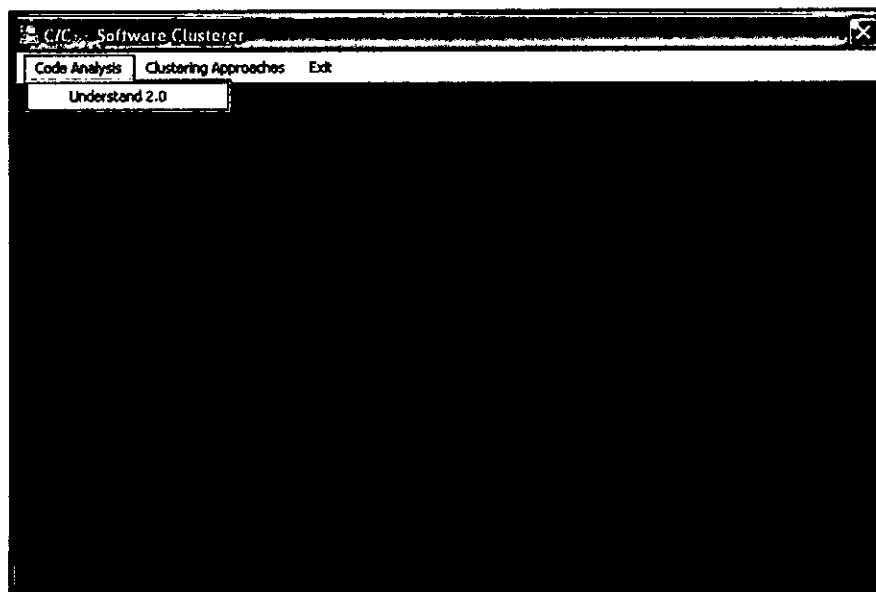


Fig. 4.2 Run of *Understand 2.0*

Step 2: After clicking on the **Understand 2.0** menu, the main interface of *Understand 2.0* is loaded with **Understand Analyst - [Getting Started]** tag as shown in Figure 4.3. By clicking on the **New Project** menu of the main interface of *Understand 2.0*, a **New Project** for source code analysis is initiated. After this, a name is given to the newly

created project. Here the name *UnderstandLibrary.ldb* is given to the newly created project.

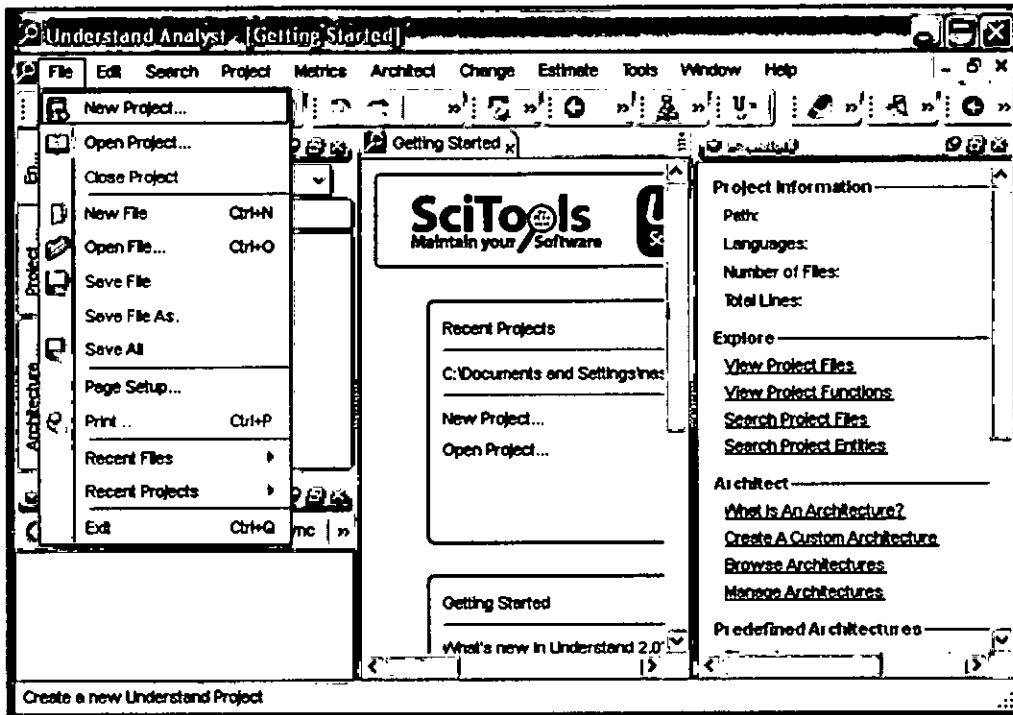


Fig. 4.3 Main window of Understand 2.0 used to create the new project

Step 3: Next, the source code file of the Library Management System (*Library.cpp*) is given to *Understand 2.0* as the input shown in Figure 4.4.

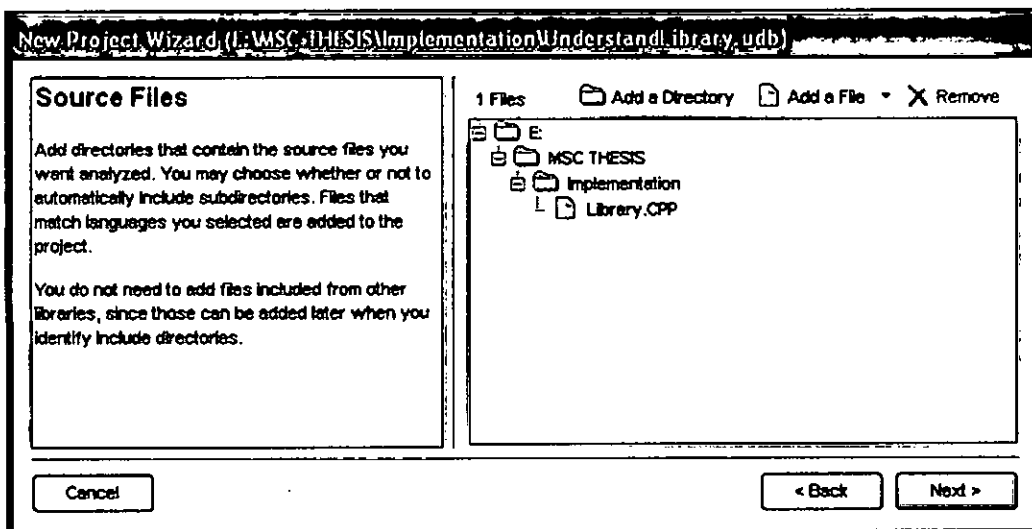


Fig. 4.4 Source Code of Library Management System as input to Understand 2.0

Step 4: After getting the source code file, *Understand 2.0* immediately provides us with a versatile analysis of the source code as shown in Figure 4.5.

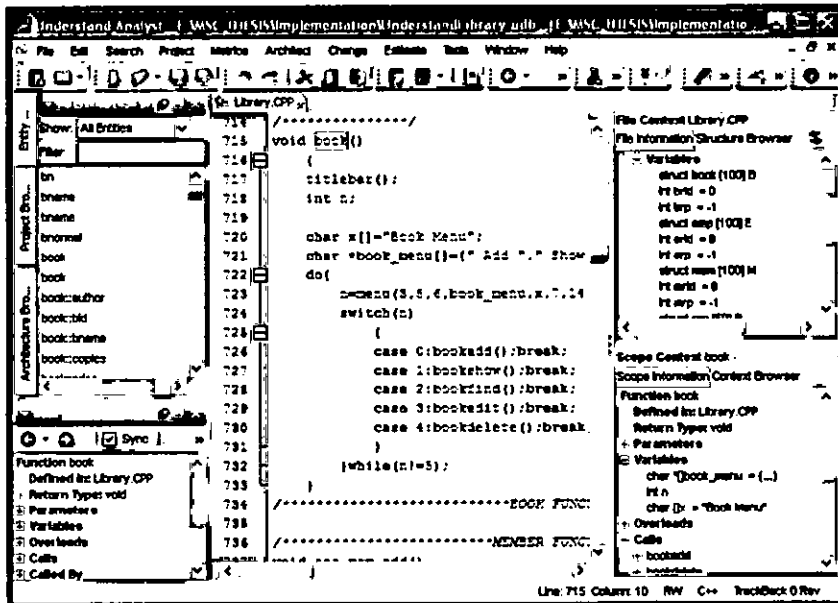


Fig. 4.5 Analysis of the Source Code

Step 5: The intermediate representation is formed by selecting *File Contents Report*, *Program Unit Cross Reference Report* and *Simple Invocation Tree Report* from the project reports provided by *Understand 2.0* as shown in Figure 4.6.

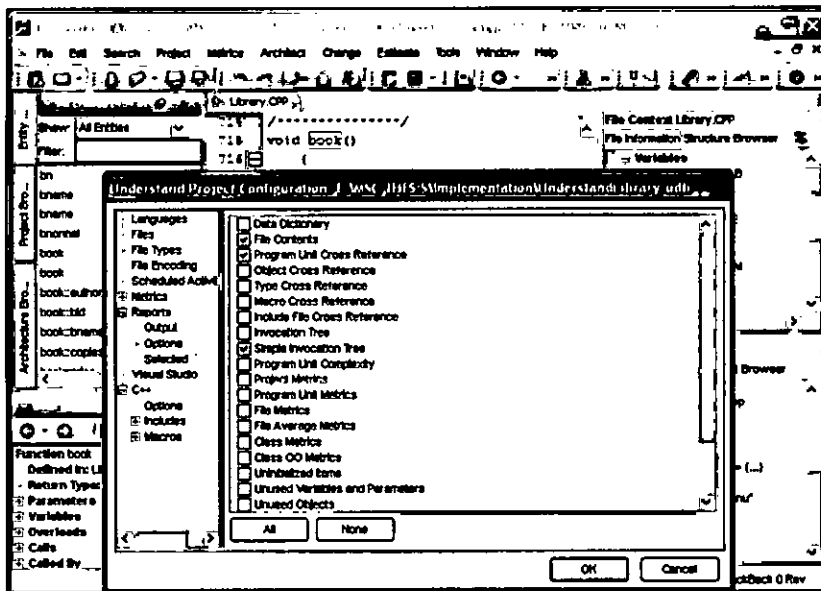


Fig. 4.6 Intermediate file generation

This intermediate representation of the source code consists of –

1. *File Contents Report*: This report contains types (structures), global variables and global functions used in the source code.
2. *Program Unit Cross Reference Report*: This report contains the calling (multiple) of a function by other functions.
3. *Simple Invocation Tree Report*: This report contains the invocation of other functions in a function.

By accomplishing these steps a Library Management System (*Library.cpp*) developed in C programming language is analyzed using *Understand 2.0* and the intermediate representation file *UnderstandLibrary.txt* is generated. The file *UnderstandLibrary.txt* contains the following in brief –

*Table 4.1 Brief content of the Intermediate Representation*

|                             |  |
|-----------------------------|--|
| <i>File Contents Report</i> | <p><b>Types (Structures):</b><br/>_ret, book, d, emp, mem, ren</p> <p><b>Global Variables:</b><br/>B, brid, brp, E, erid, erp, M, mrid, mrp, R, RD, rrp, T, trp</p> <p><b>Global Functions:</b><br/>_return, bcustom, bidavl, bidchk, binfo, bmatch, bnormal, book, bookadd, bookdelete, bookedit, bookfind, bookshow, by_addr, by_author, by_bid, by_bname, by_eaddr, by_eid, by_ename, by_mid, by_mname, by_rbid, by_rbname, by_rmid, by_rmname, control, ecustom, eidavl, eidchk, einfo, empadd, empdelete, empedit, empfind, employee, empshow, ename_eaddr, enormal, esex, info, main, mcustom, memadd, member, memdelete, memedit, memfind, memshow, menu, midavl, midchk, minfo, mnormal, name_addr, name_author, one_book_add, one_book_delete, one_book_edit, one_book_show, one_emp_add, one_emp_delete, one_emp_edit, one_emp_show, one_mem_add, one_mem_delete, one_mem_edit, one_mem_show, one_rent_add, one_rent_edit, one_rent_show, one_ret_add, one_ret_show, rbidavl, rent, rentadd, rentedit, rentfind, rentshow, retadd, retshow, rmidavl, rnormal, sex, sex_code, titlebar, tnormal</p> |
|-----------------------------|--|



|                          | <b>Callee</b>                                  | <b>Caller</b>   | <b>Line number<br/>in<br/><i>Library.cpp</i></b> |
|--------------------------|--|-----------------|--|
|                          | <i>Program Unit Cross<br/>Reference Report</i> | book (Function) | control  |
| bookadd (Function)       |  | book            | 726  |
| bookdelete (Function)    |  | book            | 730  |
| bookedit (Function)      |  | book            | 729  |
| ... ..                   |  | ...             | ...  |
| member (Function)        |  | control         | 2434   |
| ... ..                   |  | ...             | ...  |
| one_book_show (Function) |  | one_book_edit   | 1606   |
|                          |  | one_book_delete | 1705   |
| ... ..                   | ...  | ...             |  |

|  | <b>Invoked by</b> | <b>Invokees</b>  |
|--|-------------------|--|
| <i>Simple Invocation<br/>Tree Report</i> | book              | titlebar, menu, bookadd, bookshow, bookfind, bookedit, bookdelete  |
|  | bookadd           | one_book_add, printf, toupper, getche  |
|  | bookdelete        | textcolor, textbackground, titlebar, printf, getch, one_book_delete, toupper, getche                       |
|  | bookedit          | textcolor, textbackground, titlebar, printf, getch, one_book_edit, toupper                                 |
|  | bookfind          | titlebar, textcolor, textbackground, printf, getch, menu, gotoxy, by_bid, by_bname, by_author, name_author |
|  | bookshow          | titlebar, textcolor, textbackground, printf, getch, menu, bnormal, bcustom                                 |
|  | ... ..            | ... ..   |

### 4.2.2 Creation of the MDG

*Understand* 2.0 writes the intermediate representation in the file *UnderstandLibrary.txt*. Then *BUET Cluster* 1.0 parses the intermediate representation and creates an in-memory representation (based on the dependencies among the entities in the intermediate representation) of the software system which is known as *MDG*.

On the way of creating the *MDG*, the following steps are performed:

Step 1: *BUET Cluster* 1.0 is run from the menu shown in Figure 4.7 of our implemented project. The path is: **Clustering Approaches** → **BUET Cluster 1.0**.

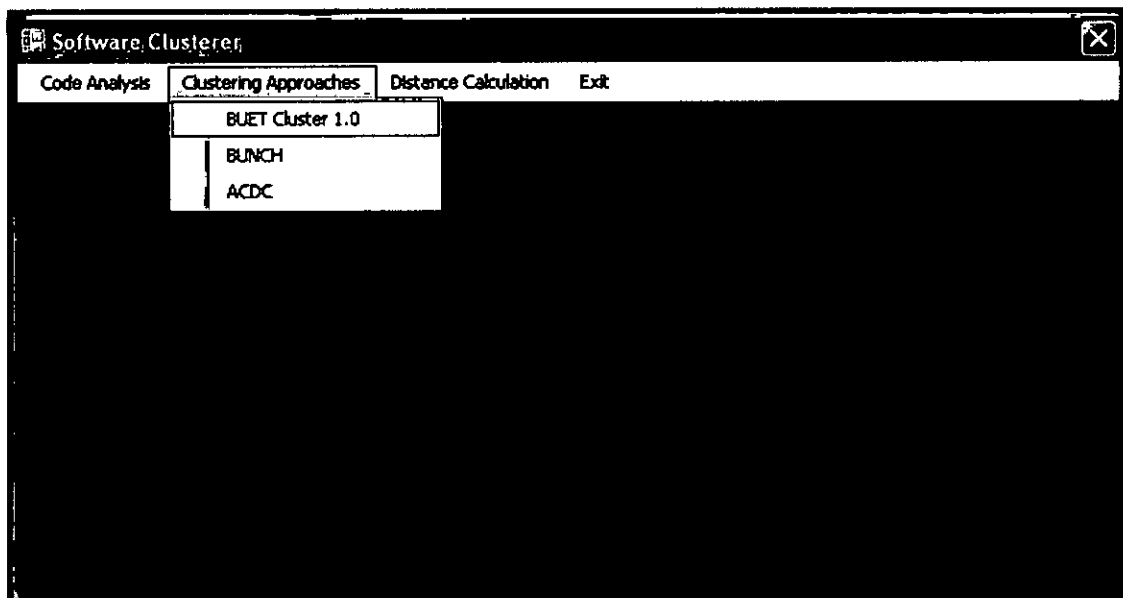


Fig. 4.7 Run of *BUET Cluster* 1.0

Step 2: After clicking on **BUET Cluster 1.0** menu, the main form shown in Figure 4.8 named **BUET Cluster 1.0** is loaded. Then, by clicking the **Load Data** Button the intermediate file *UnderstandLibrary.txt* is loaded for parsing.

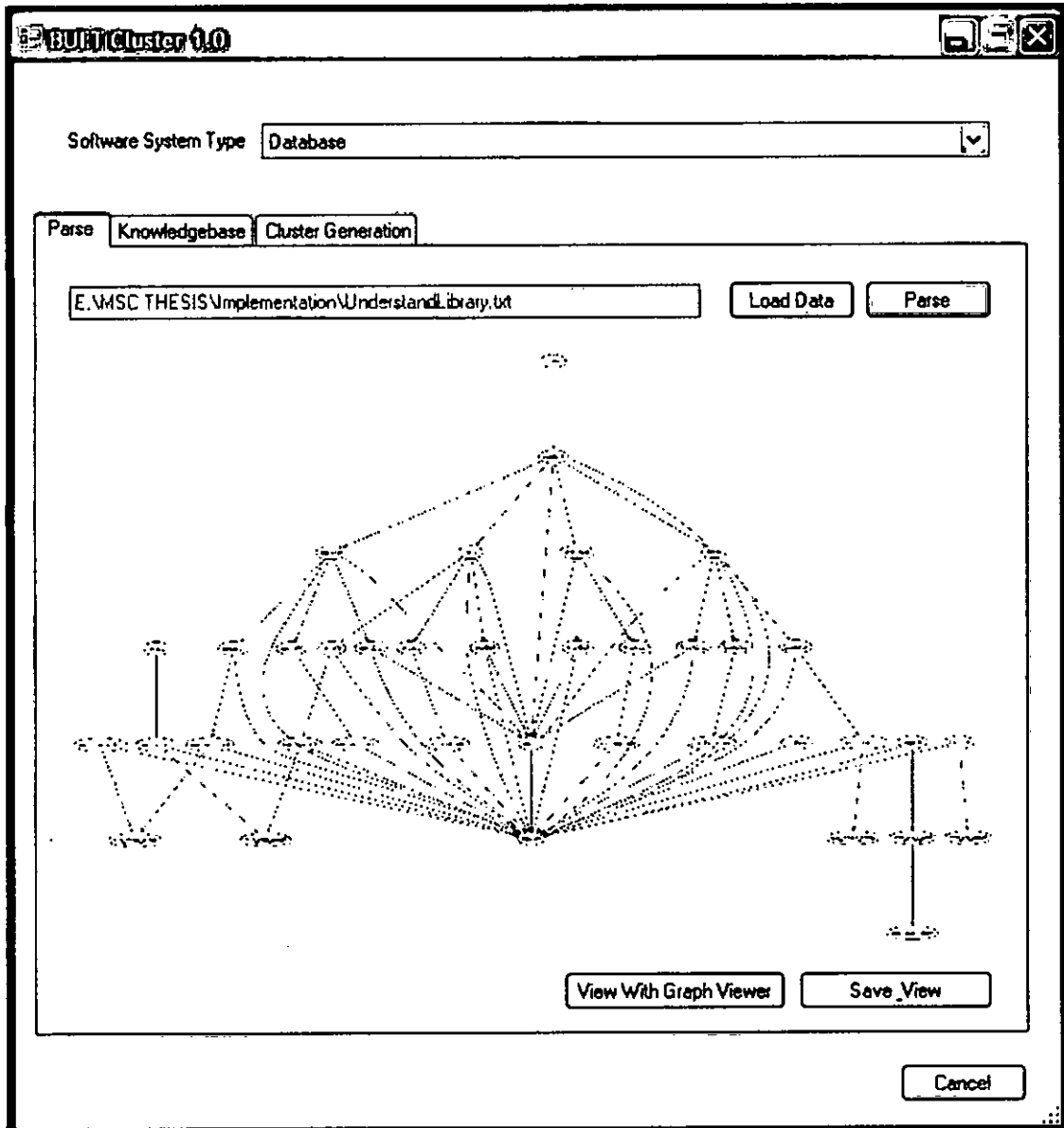


Fig. 4.8 Parsing of UnderstandLibrary.txt

When **Parse** Button is clicked, *Parse\_Data()* function takes the intermediate file *UnderstandLibrary.txt* as the input and finds out the dependencies among the entities in the intermediate representation by tracking many properties stored in the file *UnderstandLibrary.txt*. Firstly, *Parse\_Data()* function tracks types (structures), global functions and global variables. Then it tracks the usage of structures or global variables by the functions. The frequency of the usage is also determined. *Parse\_Data()* also tracks the function calls from every individual functions and the frequency of calling. Each and

every information extracted from the *UnderstandLibrary.txt* by *Parse\_Data()* is stored in the in-memory collection (a C# data type) of associated objects. After this, *Prune\_Data()* function is used to exclude some information (e.g. functions like *main*) which is insignificant and in some times detrimental to the clustering process. We call this in-memory representation as *MDG*.

#### 4.2.3 Visualization of *MDG* using *Graphviz-win v2.16*

*Graphviz-win v2.16* is a tool for creating, viewing, editing and processing *DOT* files. *DOT* is a versatile, command line graph drawing utility (layout engine) that has been used by researchers for many years. One notable feature of *DOT* is its graph description language. Users specify the nodes and edges of the graph in a text file and may add any number of optional attributes to the nodes and edges in order to control the appearance of the graph. For example, users can set attributes to control the font, font size, edge style, edge labels, colors of the nodes and edges, fill types, scaling and so on. *DOT* draws directed graphs as hierarchies. It reads attributed graph text files and produces output in any one of twelve different supported output file formats (e.g., GIF, JPEG, PostScript, etc.).

The parser of *BUET Cluster 1.0* generates the graph description language acceptable to *DOT* to view the *MDG* using the *Generate\_MDG()* function. When **Parse Button** is clicked, the *Generate\_MDG()* function writes the appropriate graph description language in the file named *MDG.txt*. A sample of *MDG* is shown in Table A.1 in Appendix A.

Now, to view the *MDG* using *Graphviz-win v2.16*, the following steps are performed:

**Step 1:** *Graphviz-win v2.16* is directly run by clicking the **View with Graph Viewer** Button of the main form and then the editor of *Graphviz-win v2.16* is loaded. After this, the *MDG.txt* file loaded to the editor as shown in Figure 4.9.

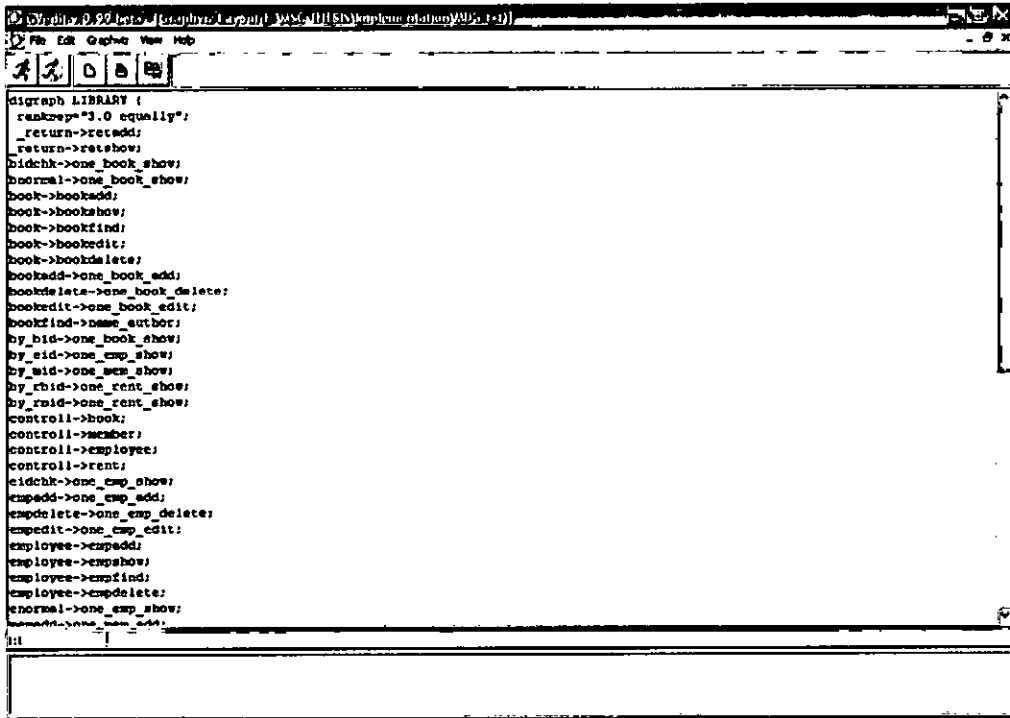


Fig. 4.9 Editor of Graphviz-win v2.16

Step 2: After getting the *MDG.txt* file, *Graphviz-win v2.16* runs *DOT* layout engine to view the *MDG* as shown in Figure 4.10.

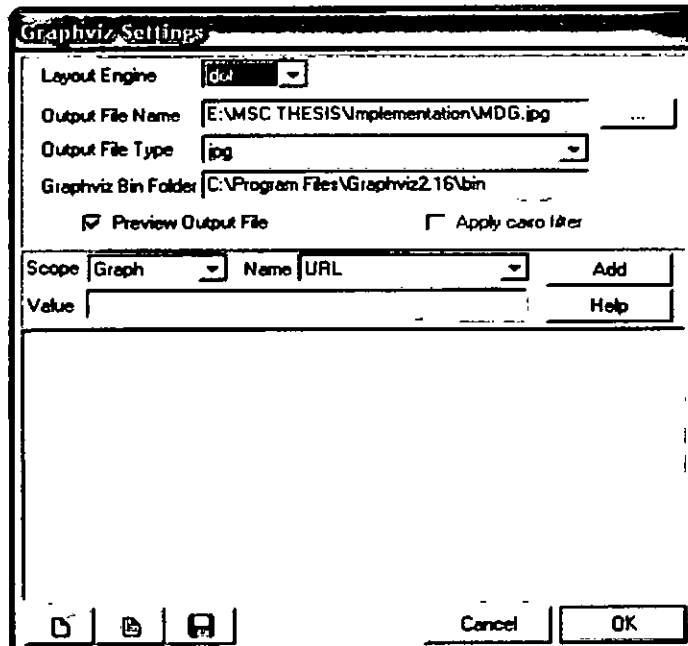


Fig. 4.10 DOT layout engine to view the MDG

After clicking the **OK** Button, we can visualize the *MDG* from the generated image file *MDG.jpg* as shown in Figure 4.11.

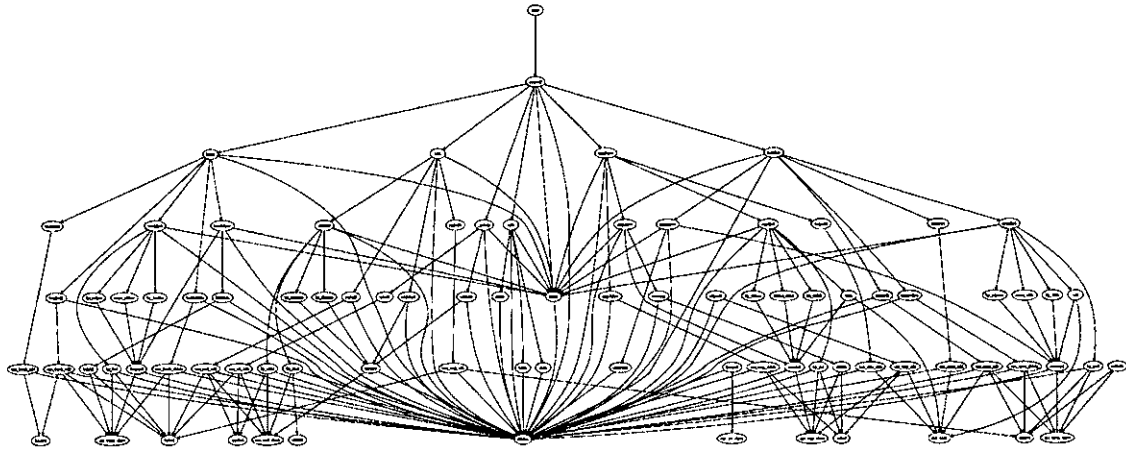


Fig. 4.11 *MDG.jpg* to view the *MDG*

Also, *BUET Cluster 1.0* can generate a preview of *MDG* in the picture box using *WINGRAPHWIZlib COM* (Component Object Model). The *WINGRAPHWIZlib COM* is integrated with *BUET Cluster 1.0* through a class named *clsGraphViz*. The code for the integration is listed as follows.

```
clsGraphViz oGraphViz = new clsGraphViz();  
pbImageMDG.Image = oGraphViz.CreateImage(MDG.txt);
```

### 4.3 Generation of the Subsystems using *BUET Cluster 1.0*

In the previous chapter, we have presented the theoretical steps of how *BUET Cluster 1.0* can utilize the Knowledgebase to generate more accurate subsystems. As we know already, the Knowledgebase acts as the repository of information about the internal structure of the Generic types of software systems to help computing an appropriate similarity matrix. Indispensable subsystems of any software system can also be obtained from the Knowledgebase. Now, we present the development and incorporation of the Knowledgebase into *BUET Cluster 1.0*.

#### 4.3.1 Knowledgebase Development

Knowledgebase development is one of the most vital steps in *BUET Cluster 1.0*. This work is done by the experts mainly. They identify the conditions for inter-relations and differences among the entities of the Generic types of the software systems by

determining the similarity measurement criteria and their respective weightages. They also figure out the indispensable subsystems (if any) of the Generic types and makes them as the part of the Knowledgebase. The Knowledgebase can also be developed from external sources of information and can be improved as a continuous process from the experience of clustering. On the way of developing the Knowledgebase in *BUET Cluster 1.0*, the following steps are performed:

Step 1: Similarity measurement criteria and their respective weightages are given as shown in Figure 4.12.

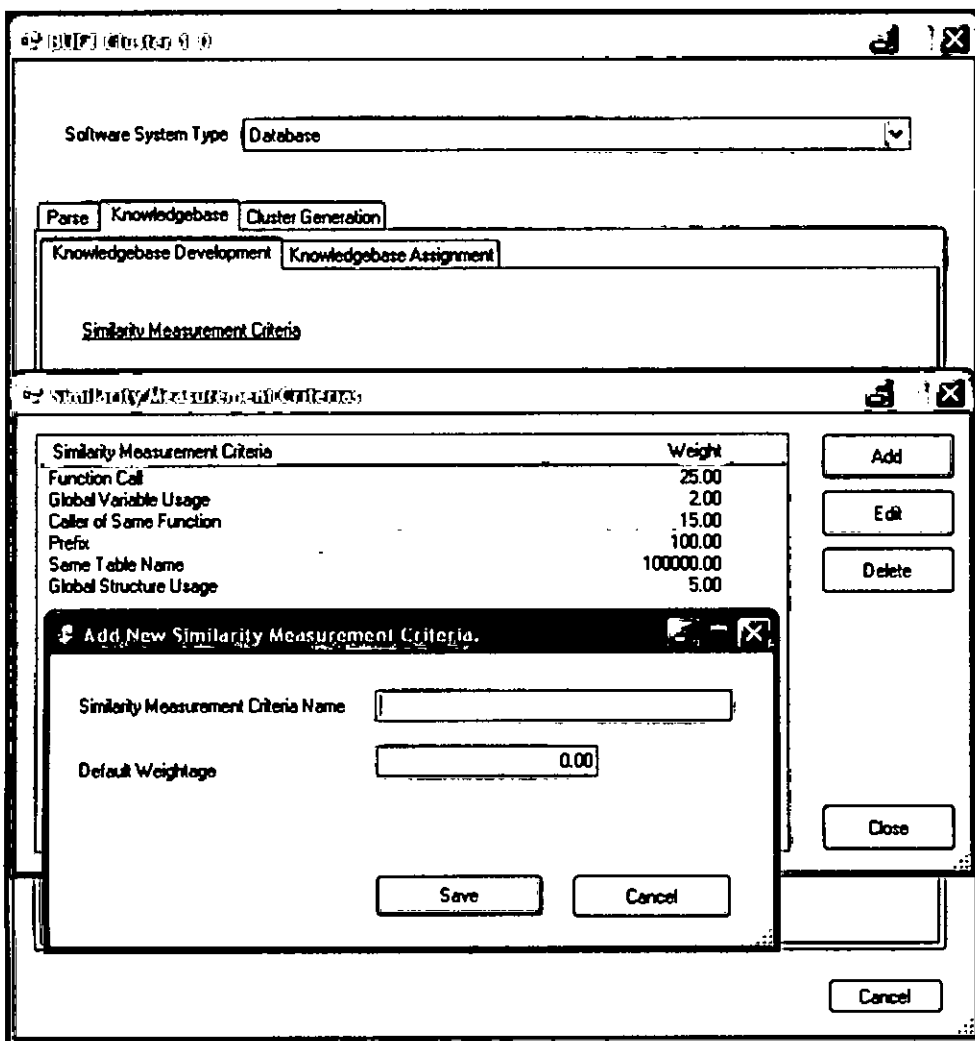


Fig. 4.12 Knowledgebase Development

Step 2: Indispensable subsystems (soul clusters) along with their property-identifying information (prefixes) are given as shown in figure 4.13.

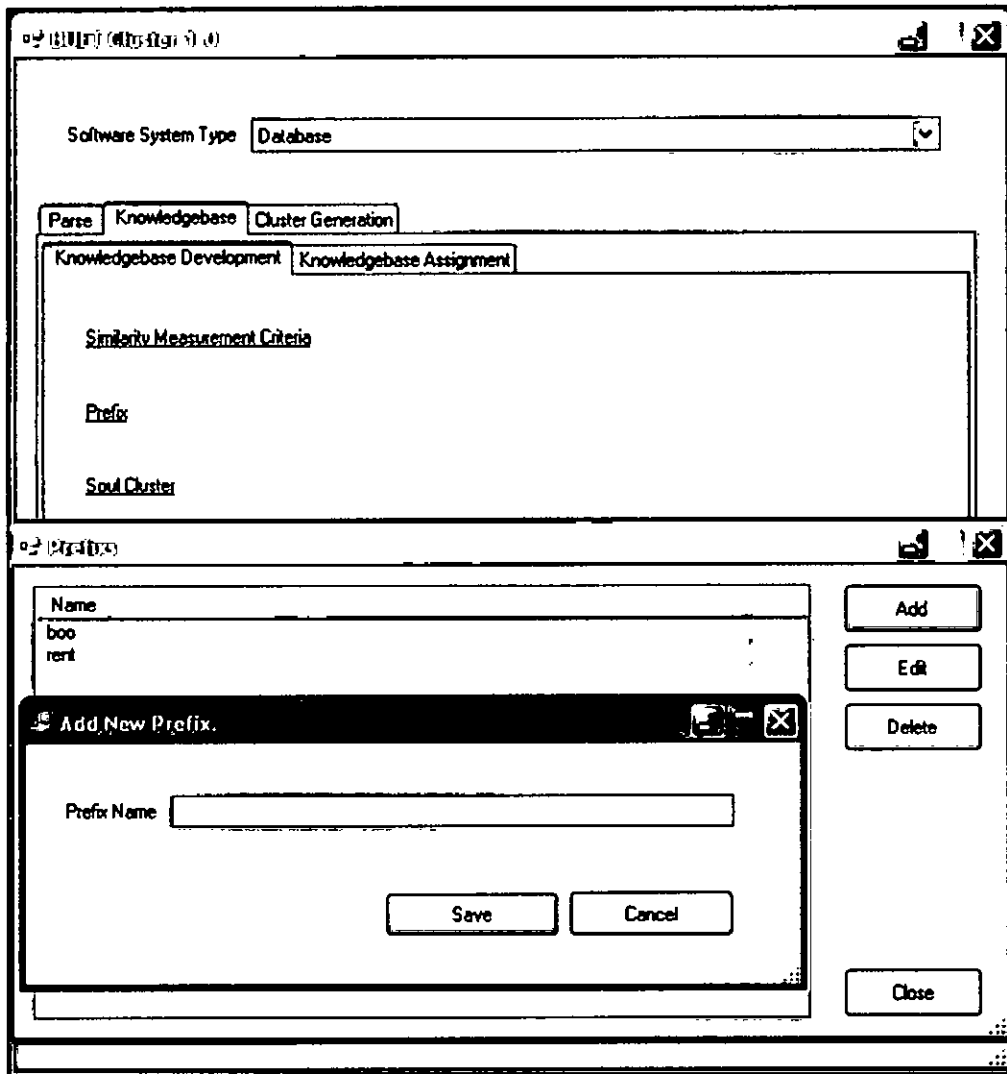


Fig. 4.13 Soul cluster identification

### 4.3.2 Knowledgebase Adaptation

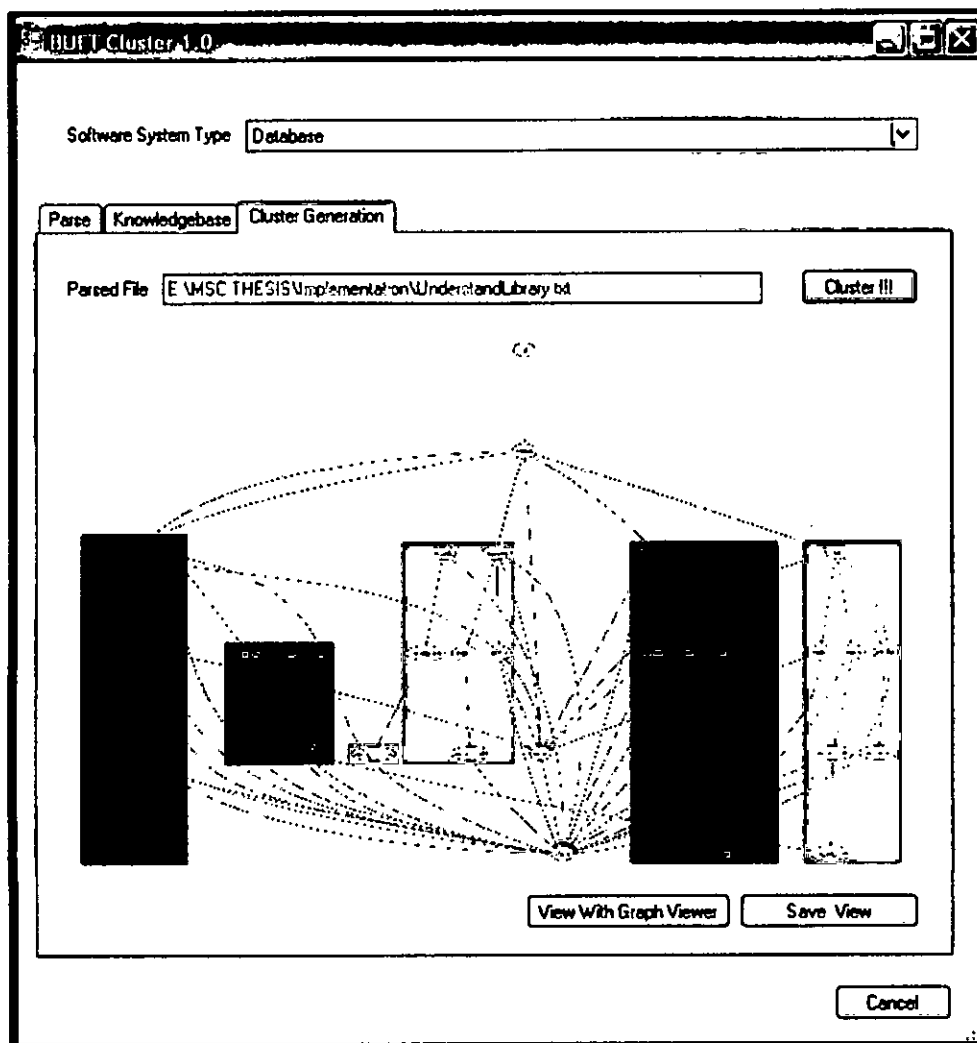
When a software system is selected for clustering, all the information (similarity measurement criteria and their respective weightages, soul clusters etc.) of the selected software system's Generic type is obtained from the Knowledgebase and used in the processes of clustering. But when the clustering result is not up to the mark using the information of the selected software system's Generic type, there is a need for adaptation.



At this point, similarity measurement criteria and their respective weightages are altered; soul clusters are added or removed in a trial and error basis. In this way, the Knowledgebase is enriched to generate more appropriate subsystems from a continuous learning process by the experience of clustering. The already demonstrated pages of the Knowledgebase development have the facility of adaptation by adding or modifying soul clusters, prefixes and similarity measurement criteria.

### 4.3.3 Cluster Generation

When **Cluster !!!** Button is pressed, clustering will be performed as per clustering technique presented in Chapter 3. The resultant subsystems are shown in Figure 4.14.



*Fig. 4.14 Resultant subsystems*

#### 4.4 Clustering software systems using *BUNCH* and *ACDC*

*BUNCH* and *ACDC* are the clustering tools intended to aid the software engineers in understanding, verifying and maintaining a source code base. These tools also let the user evaluate the quality of an application's modularization by analyzing the *MDG*.

The first step in the clustering processes using *BUNCH* or *ACDC* involves creating an *MDG* based on the source code components and relations. The next step involves the partition of the *MDG* using *BUNCH* or *ACDC*. Finally, the resulting subsystem decomposition is visualized.

*BUNCH* or *ACDC* can be run from the menu as shown in Figure 4.15.

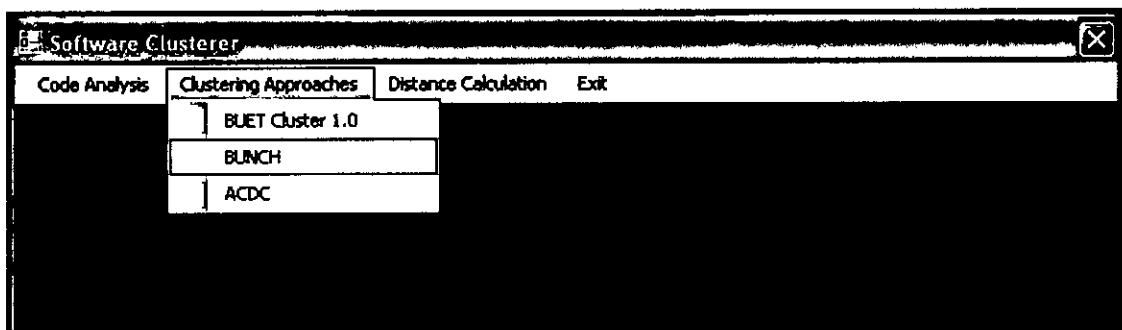


Fig. 4.15 Pathway for running *BUNCH* and *ACDC*

By clicking on *BUNCH* or *ACDC*, corresponding clustering software is involved for clustering. Please refer to [17] and [18] for details of using *BUNCH* and *ACDC* for generating clusters. Only *BUNCH* and *ACDC* are the available clustering tools available and downloadable for free of cost on the Internet. The other clustering tools require licensing for which could not be implemented due to unavailability of funds. That is why we only compare our approach with these two available tools. The following chapter presents results with analysis.

# Chapter 5

## Results with Comparison

In the previous chapter, we have used a Library Management System (*Library.cpp*) developed in C programming language as a case study. In this chapter, at first we will present, analyze and compare the resultant clusters of that Library Management System generated by *BUET Cluster 1.0*, *BUNCH* and *ACDC*. Then we will also present, analyze and compare the resultant clusters of Xfig (an open source drawing tool developed in C programming language) generated by *BUET Cluster 1.0*, *BUNCH* and *ACDC*. It is worth mentioning that *BUNCH* and *ACDC* are one of the most recognized tools in the arena of software clustering. All the results in this chapter have been generated using a Hewlett-Packard HP Compaq dq7300 Microcomputer having Intel(R) Core(TM)2 CPU 6400 @ 2.13 GHz and 0.98 GB of RAM.

### 5.1 Comparison Criteria

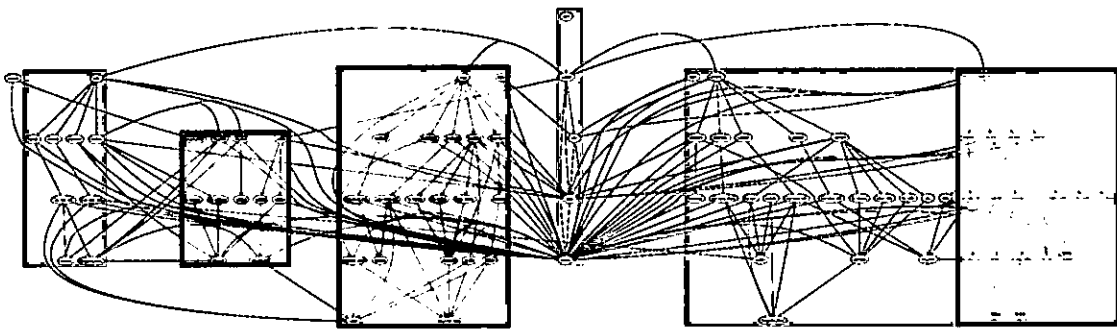
It is very tough to justify the quality of software clustering results based on one single criterion. As the result of software clustering is more related with the quality acceptable to the software engineers, it is very difficult to parameterize the result in a single numeric value. There are several characteristics in a software clustering result [19]. So, software clustering results are usually compared based on the criteria stated below:

- *Stability*: Similar clustering should be produced for similar versions of a software system.
- *Meaningfulness*: Generated clusters should resemble the subsystems of original system.
- *Extremity of Cluster Distribution*: The cluster size distribution of a clustering should not exhibit extremity. That is – the majority of software artifacts should not be grouped into one or few huge clusters and also there should not be clusters with very few software artifacts.
- *Human Intervention*: Clustering process should not require lots of human interventions.

- *Vulnerable to Error*
- *Computation Time*: The clustering should not require too long.

## 5.2 MoJo Distance for Comparison of Clustering Quality

MoJo distance [20] is the distance between two clusterings  $A$  and  $B$  of the same software system. It is defined as the minimum number of Move or Join operations one needs to perform in order to transform either  $A$  to  $B$  or vice versa. The smaller the MoJo distance between a resultant decomposition  $A$  and the *Gold Standard* decomposition  $B$ , the more effective the approach that created  $A$ . Gold Standard is defined as the expert decomposition of any software system. It is the most important reference point to which a software clustering result can be judged. Here we outline the Gold Standard decompositions for Library Management System and Xfig in Figure 5.1 and Figure 5.2 respectively.



*Fig. 5.1 Gold Standard for Library Management System (The original version of this figure is shown in Figure B.1)*

From Figure 5.1, we see the true decomposition or Gold Standard decomposition of the Library Management System has 5 (five) distinct subsystems namely “book”, “employee”, “member”, “rent” and “return”. There is another cluster containing functions not related to those subsystems. So, the total number of clusters is 6 (six).

The figure showing the clustering results of a software system generally does not fit in a small area. In this chapter, the Gold Standard decompositions and the clustering results of the software systems are shown using scaled version of the original figures. Original figures are shown in Appendix B.



Fig. 5.2 Gold Standard for Xfig (The original version of this figure is shown in Figure B.2)

From Figure 5.2, we see the true decomposition or Gold Standard decomposition of Xfig has 2 (two) distinct subsystems [21]. One subsystem takes specification of objects and another subsystem renders the object.

### 5.3 Resultant Clusters of the Library Management System

#### 5.3.1 Results generated by BUET Cluster 1.0

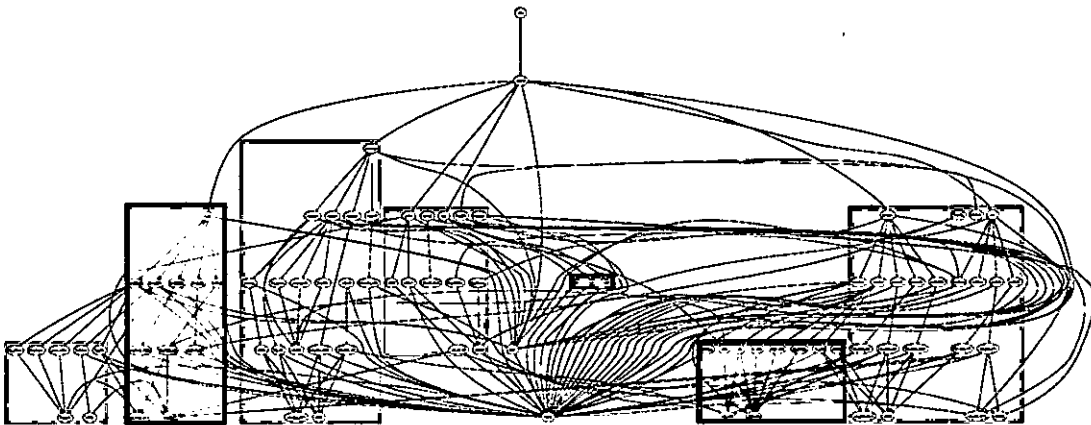
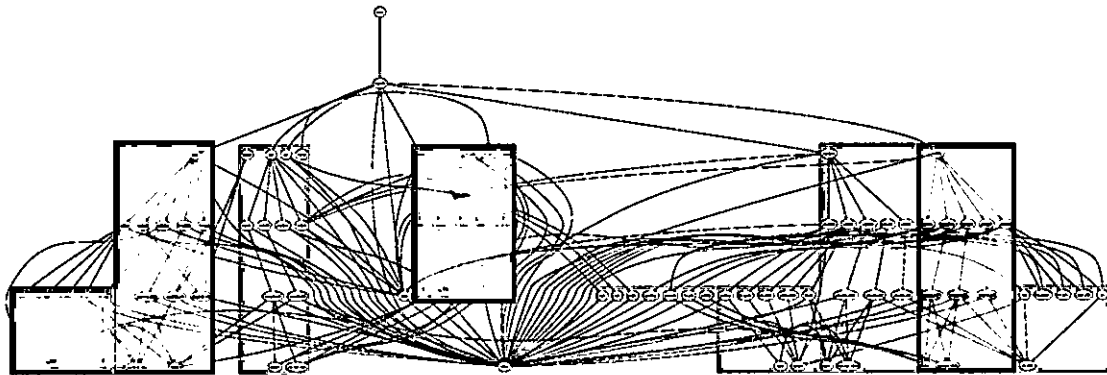


Fig. 5.3 Resultant Clusters from BUET Cluster 1.0 (with threshold 8)

Figure 5.3 and Figure 5.4 presents the results of clustering of the Library Management System using BUET Cluster 1.0 using different values of threshold. We see BUET Cluster 1.0 has clustered the Library Management System (*Library.cpp*) quite fairly using threshold value 8 and 9. Here the subsystem “Book” was supplied from the Knowledgebase as an integral part of the Library Management System. However, BUET Cluster 1.0 finds other subsystems of the Library Management System like “employee”,

“member”, “rent”, “return” and others automatically and combines them with the subsystem supplied from the Knowledgebase (“Book”) to generate the final result.

One major characteristics of the result using threshold value 8 is: the subsystem “employee” is almost flawless as it contains almost every functions belonging to “employee” of Gold Standard.



*Fig. 5.4 Resultant Clusters from BUET Cluster 1.0 (with threshold 9)*

The only difference between the two results generated by using different threshold is that the later result splits the subsystem “employee”. Some of its functions have been placed in the extra cluster of the result. Thus it is obvious that selection of threshold is an important factor in the Software Clustering. The Software Engineers responsible for clustering or reengineering must select a suitable threshold value for effective clustering on trial and error basis.

*Overall quality of results generated by BUET Cluster 1.0:* The resultant clusters generated by *BUET Cluster 1.0* contain some anomalies as many of the functions have not been placed in the most appropriate clusters. However the result as a whole does not contradict with the meaningfulness, as we see that most of the subsystems figured out can affirm their own identity.

The results shown in Figure 5.3 and Figure 5.4 are not necessarily the best one generated by *BUET Cluster 1.0*. Yet one can find more suitable combination of similarity measurement criteria and their respective weightages for the Library Management System and improve the clustering result.

### 5.3.2 Results generated by *BUNCH*

As we already know, *BUNCH* (one of the most recognized tool for software clustering) uses heuristic approach and therefore we get more than one clustering result from the Library Management System for different runs of this tool.

Here, we have generated three resultant clusters from three different runs of *BUNCH* on the same Library Management System. Now we present those results in detail –

*Result 1 from BUNCH:*

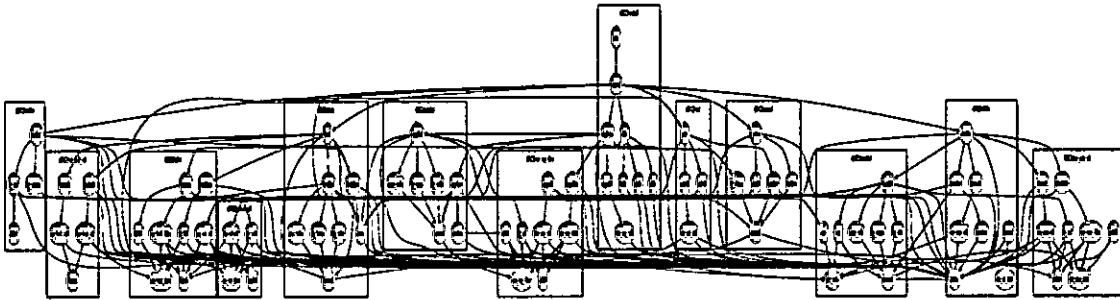


Fig. 5.5 Result 1

From Figure 5.5 we see, *BUNCH* has created 13 clusters in Result 1. From Result 1 we see *BUNCH* finds “book” and “member” subsystems to some extent. But other subsystems like “employee”, “rent” and “return” got divided and amalgamated with each other. Hence the quality of the result is degraded substantially.

*Result 2 from BUNCH:*

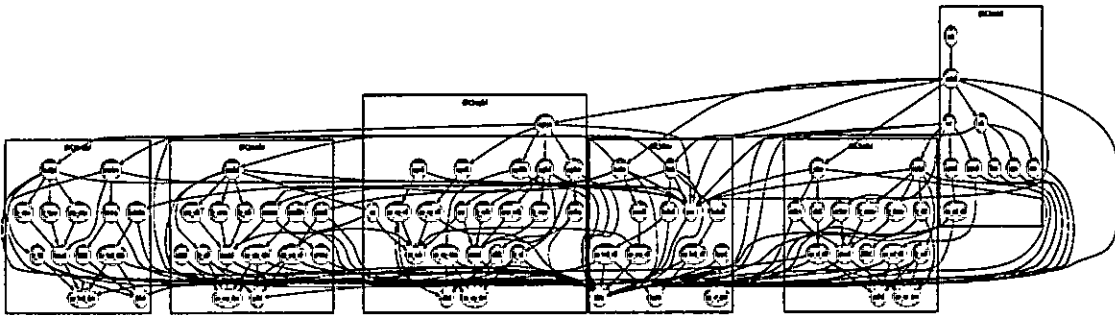
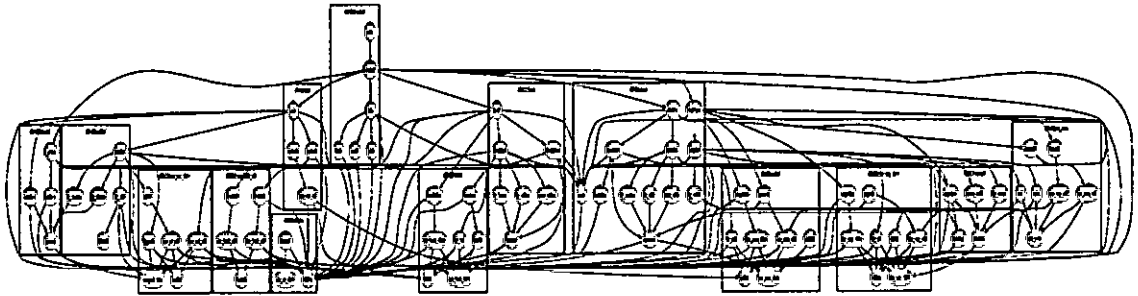


Fig. 5.6 Result 2

From Figure 5.6 we see, *BUNCH* has created 6 clusters in Result 2. From Result 2 we see *BUNCH* has discovered the “employee” subsystem perfectly but inter-mixes other vital subsystems like “book” and “member”, which has made the total result less meaningful.

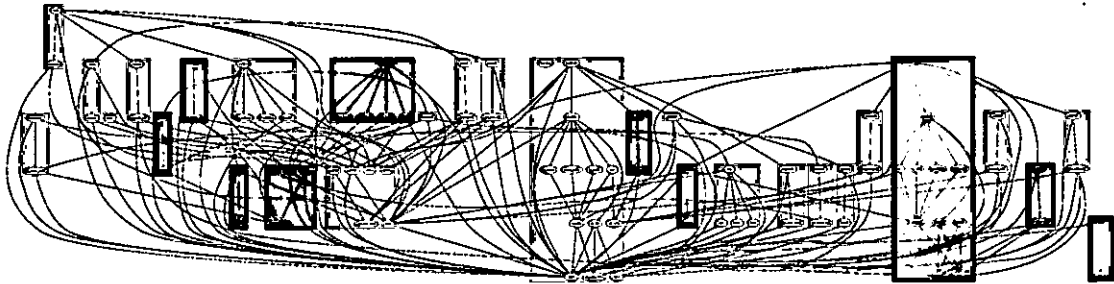
*Result 3 from BUNCH:*



*Fig. 5.7 Result 3*

From Figure 5.7 we see, *BUNCH* has created 14 clusters in Result 3. From Result 3 we see *BUNCH* has found “book”, “rent” and “return” subsystems partially but inter-mixed other vital subsystems like “member” and “employee”, which has made the total result less meaningful.

### 5.3.3 Result generated by *ACDC*



*Fig. 5.8 Resultant Clusters from ACDC*

From Figure 5.8 we see, *ACDC* has created 29 clusters. Thus, the result contains many small clusters with very few software artifacts. From the result we see *ACDC* has found “employee” and “member” quite fairly. But other subsystems like “book”, “rent” and “return” was not identifiable as they got inter-mixed and broke into pieces.



## 5.4 Resultant Clusters of the Xfig

### 5.4.1 Results generated by *BUET Cluster 1.0*

Here we consider a range of threshold values (from 2 to 10) for clustering Xfig. The results are shown in Table 5.1.

Table 5.1 Results from *BUET Cluster 1.0* for Xfig for different threshold values

| Results  | Threshold | Computation Time<br>(in seconds) | MoJo Distance |
|----------|-----------|----------------------------------|---------------|
| Result 1 | 2         | 47.306                           | 4             |
| Result 2 | 3         | 44.041                           | 5             |
| Result 3 | 4         | 41.945                           | 6             |
| Result 4 | 5         | 39.277                           | 7             |
| Result 5 | 6         | 35.906                           | 8             |
| Result 6 | 7         | 33.339                           | 9             |
| Result 7 | 8         | 30.875                           | 10            |
| Result 8 | 9         | 29.017                           | 11            |
| Result 9 | 10        | 26.944                           | 12            |
| Average: | 6         | 36.517                           | 8             |

From the results shown in Table 5.1, we find that *BUET Cluster 1.0* generates the best result for the threshold value 2. The subsystems generated by *BUET Cluster 1.0* using threshold value 2 is shown in Figure 5.9.

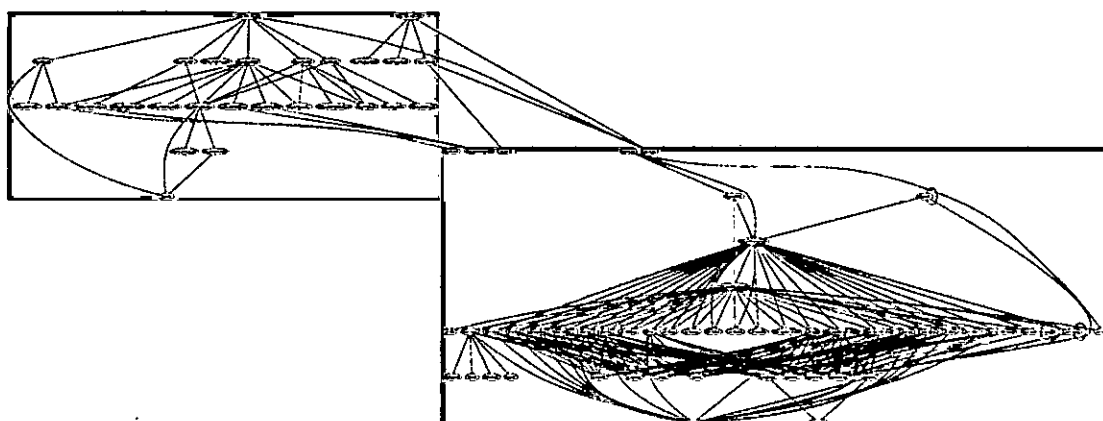


Fig. 5.9 Resultant Clusters from *BUET Cluster 1.0* using threshold value 2 (The original version of this figure is shown in Figure B.3)

From Figure 5.9 we see *BUET Cluster 1.0* has clustered Xfig quite fairly. The result contains two distinct subsystems of Xfig with very little anomaly.

### 5.4.2 Results generated by *BUNCH*

We run *BUNCH* for multiple times for clustering Xfig. The results are shown in Table 5.2.

Table 5.2 Results from *BUNCH* for Xfig

| Results         | Number of Clusters | Computation Time (in seconds) | MoJo Distance |
|-----------------|--------------------|-------------------------------|---------------|
| Result 1        | 8                  | 0.105                         | 6             |
| Result 2        | 8                  | 0.125                         | 6             |
| Result 3        | 9                  | 0.109                         | 6             |
| Result 4        | 7                  | 0.093                         | 6             |
| Result 5        | 8                  | 0.098                         | 6             |
| Result 6        | 7                  | 0.109                         | 6             |
| Result 7        | 8                  | 0.094                         | 6             |
| Result 8        | 8                  | 0.103                         | 6             |
| Result 9        | 8                  | 0.094                         | 6             |
| <b>Average:</b> | <b>7.89</b>        | <b>0.103</b>                  | <b>6</b>      |

From the results shown in Table 5.2, we find that *BUNCH* generates almost similar types of results in every run. The best MoJo distance here is 6. The average number of clusters is 7.89. We present a result with 8 clusters in Figure 5.9.

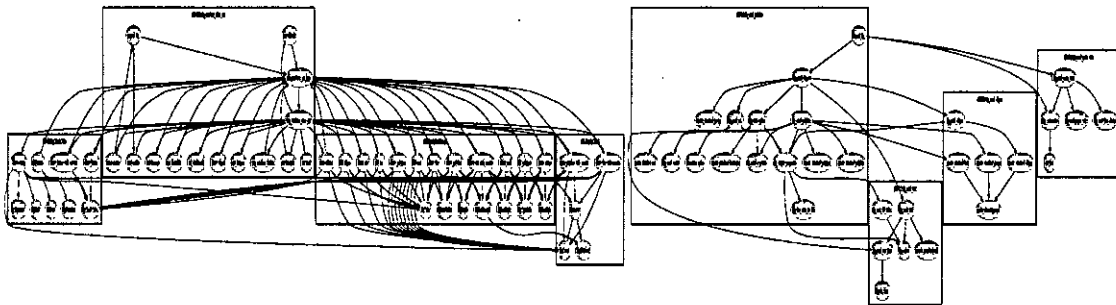


Fig. 5.10 Resultant Clusters from *BUNCH* (The original version of this figure is shown in Figure B.4)

From Figure 5.10 we see *BUNCH* has clustered Xfig quite fairly but the result contains more subsystems than necessary.

### 5.4.3 Result generated by *ACDC*

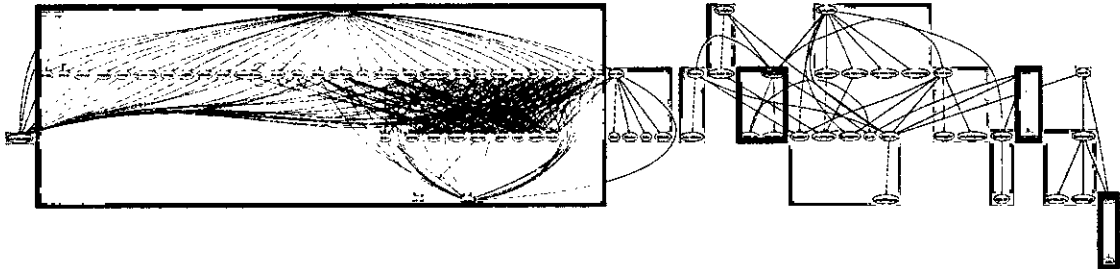


Fig. 5.11 Resultant Clusters from *ACDC* (The original version of this figure is shown in Figure B.5)

From Figure 5.11 we see *ACDC* has clustered Xfig in a moderate way. *ACDC* has figured out one subsystem fairly but other subsystem has been broken into pieces.

## 5.5 Comparison among the results of *BUET Cluster 1.0*, *BUNCH* and *ACDC*

The detailed comparison is shown in Table 5.3.

Table 5.3 Comparison among *ACDC*, *BUNCH* and *BUET Cluster 1.0*

| Criteria                                 | <i>ACDC</i> |   | <i>BUNCH</i>  |  | <i>BUET Cluster 1.0</i> |   |
|--|-------------|---|---|--|-------------------------|---|
| <b>Stability</b>                         | High        | Generates identical clustering in every run.                    | Low   | Generates new clustering in every run. | High                    | Generates identical clustering in every run.                    |
| <b>Meaningfulness</b>                    | Low         | As described above.   | Medium /Low (due to local minima )                              | As described above.                    | High                    | As described above.   |
| <b>Extremity of Cluster Distribution</b> | High        | Generates too many clusters if too many patterns are discovered | Low (Because it can penalize excessive inter-cluster coupling ) | Generates subsystems of uniform size.  | Medium                  | Sometimes generates very small or comparatively large clusters. |
| <b>Human Intervention</b>                | Low         | No human intervention.  | Low   | No human intervention.                 | Medium/ Low             | Minimum human intervention.                                     |

| Vulnerable to Error     | High | As described above.   | Medium      | As described above.   | Low         | As described above.   |
|-------------------------|------|---|-------------|---|-------------|---|
| <b>Computation Time</b> | Low  | For Library Management System: 0.015 sec.<br><br>For Xfig: 0.028 sec. | Low         | For Library Management System of the three runs: $(0.125+0.067+0.093) / 3 = 0.095$ sec.<br><br>For Xfig: 0.103 sec. (avg.)  | Medium/High | For Library Management System (with threshold 8 and 9): $(14.88+12.26)/2 = 13.57$ sec.<br><br>For Xfig: 36.517 sec. (avg.)  |
| <b>MoJo distance</b>    | High | For Library Management System: 32<br><br>For Xfig: 12                 | Medium/High | Average MoJo distance for Library Management System of the three runs: $(26+30+27) / 3 = 27.67$ .<br><br>For Xfig: 6 (avg.) | Medium/Low  | For Library Management System (with threshold 8 and 9): $(9+10) = 9.5$<br><br>For Xfig for threshold value 2: 4 (The Lowest MoJo Distance)<br>Average MoJo Distance for Xfig: 8 |

### 5.6 Comparison with other Clustering Approaches

As all the clustering tools are not available for clustering we cannot analyze those techniques by comparing the results. The following table compares the characteristics of those techniques with our proposed technique:

107528

*Table 5.4 Characteristic observation*

| <b>Criteria</b>                          | <b>Rough Set Clustering</b>               | <b>LIMBO</b>   | <b>BUET Cluster 1.0</b> |
|--|---|--|-------------------------|
| <b>Stability</b>                         | Medium (for excessive human intervention) | Low (final clustering depends on distance calculation) | High                    |
| <b>Meaningfulness</b>                    | High                                      | Medium/ Low (depends on too many threshold values)     | High                    |
| <b>Extremity of Cluster Distribution</b> | Medium/ Low                               | High   | Medium                  |
| <b>Human Intervention</b>                | High                                      | Medium/ Low  | Medium/ Low             |
| <b>Vulnerable to Error</b>               | High (if human direction goes wrong)      | Medium   | Low                     |
| <b>Computation</b>                       | High                                      | Medium   | Medium/ High            |

# Chapter 6

## Conclusions

Many software engineers have noted that maintaining and understanding the structure of source code is getting harder because the size and complexity of modern software systems is growing. This coupled with associated problems such as lack of accurate design documentation and the limited availability of the original designers of the system adds further difficulty to the job of software engineers to understand the structure of large and complex software systems. The application of clustering techniques appears to be a promising way to help software engineers by enabling them to create abstract views of the system structure. This chapter summarizes the major contributions of this thesis on software clustering and presents suggestions for future research.

### 6.1 Major Contributions

In this section we highlight some of the significant research contributions of the work described in this thesis:

- *Software Clustering using the Knowledgebase*: We have introduced a novel approach for software clustering approach which can exploit the strength of the Knowledgebase to generate meaningful and legitimate subsystems. Our understanding is – **The richer Knowledgebase you have, the more accurate subsystems you get.**
- *Design and Implementation of BUET Cluster 1.0*: We have developed *BUET Cluster 1.0* as a complete tool for software clustering. This software is ready for the use of clustering of any software if the source code is available.
- *Comparison of BUET Cluster 1.0 with other Clustering Techniques*: The other clustering techniques are compared with our new clustering technique by applying both the techniques on the renowned software code available as open source. With the help of a rich Knowledgebase *BUET Cluster 1.0* is capable to outperform any available clustering tool.

## 6.2 Future Research Directions

In this section we describe some suggestions for extending our research and propose some opportunities for future research directions.

- *Improving the Knowledgebase:* The enrichment of the Knowledgebase used in software clustering is a continuous process. The Knowledgebase can be improved in a disciplined way so that the whole software engineer community can be benefited from it.
- *Introducing Multi-Layered view in the Clustering Result:* Software clustering results can be multi-layered. Software engineers should be able to drill down from the top-level subsystems to view their inner structures.
- *Improving the Computational Time:* Computational overhead in our approach is much compared to other approaches. Researchers can contribute in this area.
- *Introducing Dynamic Information in Software Clustering:* Dynamic information like calling of functions in run-time should be used in clustering process.
- *Improving Visualization service for Results with Multi-Layer view:* There should be easy ways to visualize the multi-layered results of software clustering.

## References

- [1] Tzerpos, V. and Holt, R. C., "ACDC: An algorithm for comprehension-driven clustering," *Proceedings of the 7<sup>th</sup> Working Conference on Reverse Engineering (WCRE '00)*, pp. 258–267, 2000.
- [2] Andritsos, P. and Tzerpos, V., "Information\_Theoretic Software Clustering," *IEEE Transactions on Software Engineering*, Vol-31, No. 2, pp. 150–165, 2005.
- [3] Mitchell, B.S., "A Heuristic Approach to Solving the Software Clustering Problem," *Proceedings of the International Conference on Software Maintenance (ICSM '03)*, pp. 285–288, 2003.
- [4] Jahnke, J. H. and Bychkov, Y., "Reverse Engineering Software Architecture using Rough Clusters," *Proceedings of the International Conference on Software and Knowledge Engineering (SEKE '04)*, pp. 749–757, 2004.
- [5] Hutchens, D. and Basili, R., "System Structure Analysis: Clustering with Data Bindings," *IEEE Transactions on Software Engineering*, Vol-11, No. 6, pp. 150–165, 2005.
- [6] Anquetil, N. and Lethbridge, T., "Recovering Software Architecture from the Names of the Source Files," *Journal of Software Maintenance*, Vol-11, No. 3, pp. 201–221, 1999.
- [7] Schwanke, R., "An Intelligent Tool for Re-engineering Software Modularity," *Proceedings of the 13<sup>th</sup> International Conference on Software Engineering (ICSE '91)*, pp. 321–327, 1991.
- [8] Slonim, N. and Tishby, N., "Agglomerative Information Bottleneck," *Proceedings of the Conference on Neural Information Processing Systems (NIPS-12)*, pp. 617–623, 1999.
- [9] Xiao, C. and Tzerpos, V., "Software Clustering based on Dynamic Dependencies," *Proceedings of the 9<sup>th</sup> European Conference on Software Maintenance and Reengineering (CSMR '05)*, pp. 124–133, 2005.
- [10] Maqbool, O. and Babri, H.A., "The Weighted Combined Algorithm: A Linkage Algorithm for Software Clustering," *Proceedings of the International Conference on Software Maintenance (ICSM '04)*, pp. 15–24, 2004.



- [11] Jaccard, P., "The Distribution of the Flora in the Alpine Zone," *The New Phytologist*, Vol-11, No.2, 29<sup>th</sup> February, 1912.
- [12] Hand, D., Mannila, H., and Smyth, P., "Principles of Data Mining," *The MIT Press*, 2001, ISBN 0-262-08290-X.
- [13] Townsend, C., "Introduction to Turbo Prolog," *Revised Edition*, 2000, ISBN 81-7029-104-6.
- [14] Rich, E. and Knight, K., "Artificial Intelligence," *TATA McGRAW-HILL Edition*, 1991, ISBN 0-07-460081-8.
- [15] Understand 2.0 – [www.scitools.com](http://www.scitools.com) (Last visit: 15/08/2009)
- [16] GraphViz-win v2.16 – [www.graphviz.org](http://www.graphviz.org) (Last visit: 25/09/2008)
- [17] Mitchell, B. S., "A Heuristic Search Approach to Solving the Software Clustering Problem," *Ph. D. Thesis, Drexel University*, March 2002.
- [18] Tzerpos, V., "Comprehension-Driven Software Clustering," *Ph. D. Thesis, Graduate Department of Computer Science, University of Toronto*, 2001.
- [19] Wen, Z. and Tzerpos, V., "An effectiveness measure for software clustering algorithms," *Proceedings of the 12<sup>th</sup> IEEE International Workshop on Program Comprehension (IWPC '04)*, pp. 194–203, 2004.
- [20] Tzerpos, V. and Holt, R. C., "MoJo: A Distance Metric for Software Clustering," *Proceedings of the 6<sup>th</sup> Working Conference on Reverse Engineering (WCRE '99)*, pp. 187–193, 1999.
- [21] Xfig (open source drawing tool) – [www.xfig.org](http://www.xfig.org) (Last visit: 14/07/2009)
- [22] Wikipedia – [www.wikipedia.org](http://www.wikipedia.org) ([http://en.wikipedia.org/wiki/Cluster\\_analysis](http://en.wikipedia.org/wiki/Cluster_analysis)) (Last visit: 10/07/2009)

## Appendix A

*Table A.1 Sample of MDG*

| Graph Definition Language to Visualize<br>MDG  | Remarks   |
|--|---|
| <pre> digraph LIBRARY {   ranksep="3.0 equally";   _return-&gt;titlebar;   _return-&gt;menu;   _return-&gt;retadd;   _return-&gt;retshow;   bcustom-&gt;titlebar;   bcustom-&gt;bidchk;   bidchk-&gt;bidavl;   bidchk-&gt;one_book_show;   binfo-&gt;titlebar;   bnormal-&gt;titlebar;   bnormal-&gt;one_book_show;   book-&gt;titlebar;   book-&gt;menu;   book-&gt;bookadd;   book-&gt;bookshow;   book-&gt;bookfind;   book-&gt;bookedit;   book-&gt;bookdelete;   bookadd-&gt;one_book_add;   bookdelete-&gt;titlebar;   bookdelete-&gt;one_book_delete;   bookedit-&gt;titlebar;   bookedit-&gt;one_book_edit;   bookfind-&gt;titlebar;   bookfind-&gt;menu;   bookfind-&gt;by_bid;   bookfind-&gt;by_bname;   bookfind-&gt;by_author;   bookfind-&gt;name_author;   bookshow-&gt;titlebar;   bookshow-&gt;menu;   bookshow-&gt;bnormal;   bookshow-&gt;bcustom;   by_addr-&gt;mnormal;   by_author-&gt;bnormal;   by_bid-&gt;titlebar;   by_bid-&gt;bidavl;   by_bid-&gt;one_book_show;   by_bname-&gt;bnormal;   by_eaddr-&gt;enormal;   by_eid-&gt;titlebar;   by_eid-&gt;eidavl;   by_eid-&gt;one_emp_show;   by_ename-&gt;enormal;   by_mid-&gt;titlebar; </pre> | <p>digraph LIBRARY: Name of the Graph</p> <p>ranksep="3.0 equally": Equal separation between ranks (layers of the graph), in inches.</p> <p>book-&gt;bookadd;: Edge from book to bookadd i.e. book calls bookadd.</p> |

```

by_mid->midavl;
by_mid->one_mem_show;
by_mname->mnormal;
by_rbid->titlebar;
by_rbid->rbidavl;
by_rbid->one_rent_show;
by_rbname->rnormal;
by_rmid->titlebar;
by_rmid->rmidavl;
by_rmid->one_rent_show;
by_rmname->rnormal;
controll->titlebar;
controll->menu;
controll->book;
controll->member;
controll->employee;
controll->rent;
controll->_return;
controll->_info;
ecustom->titlebar;
ecustom->eidchk;
eidchk->eidavl;
eidchk->one_emp_show;
einfo->titlebar;
empadd->one_emp_add;
empdelete->titlebar;
empdelete->one_emp_delete;
empedit->titlebar;
empedit->one_emp_edit;
empfind->titlebar;
empfind->menu;
empfind->by_eid;
empfind->by_ename;
empfind->by_eaddr;
empfind->esex;
empfind->ename_eaddr;
employee->titlebar;
employee->menu;
employee->empadd;
employee->empshow;
employee->empfind;
employee->empdelete;
empshow->titlebar;
empshow->menu;
empshow->enormal;
empshow->ecustom;
ename_eaddr->enormal;
enormal->titlebar;
enormal->one_emp_show;
esex->sex_code;
esex->enormal;
info->titlebar;
info->menu;
info->binfo;
info->minfo;
info->einfo;
main->controll;

```

```

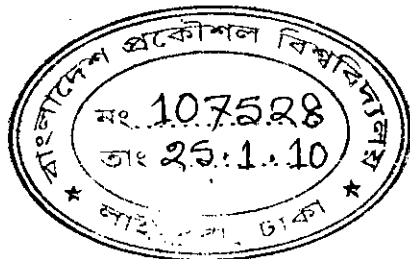
mcustom->titlebar;
memadd->one_mem_add;
member->titlebar;
member->menu;
member->memadd;
member->memshow;
member->memfind;
member->memedit;
member->memdelete;
memdelete->titlebar;
memdelete->one_mem_delete;
memedit->titlebar;
memedit->one_mem_edit;
memfind->titlebar;
memfind->menu;
memfind->by_mid;
memfind->by_mname;
memfind->by_addr;
memfind->sex;
memfind->name_addr;
memshow->titlebar;
memshow->menu;
memshow->mnormal;
memshow->mcustom;
menu->titlebar;
midchk->midavl;
midchk->one_mem_show;
minfo->titlebar;
mnormal->titlebar;
mnormal->one_mem_show;
name_addr->mnormal;
name_author->bnormal;
one_book_add->titlebar;
one_book_add->bmatch;
one_book_delete->titlebar;
one_book_delete->bidavl;
one_book_delete->one_book_show;
one_book_edit->titlebar;
one_book_edit->bidavl;
one_book_edit->one_book_show;
one_book_edit->bmatch;
one_emp_add->titlebar;
one_emp_add->sex_code;
one_emp_delete->titlebar;
one_emp_delete->eidavl;
one_emp_delete->one_emp_show;
one_emp_edit->eidavl;
one_emp_edit->one_emp_show;
one_emp_edit->sex_code;
one_mem_add->titlebar;
one_mem_add->sex_code;
one_mem_delete->titlebar;
one_mem_delete->midavl;
one_mem_delete->one_mem_show;
one_mem_edit->titlebar;
one_mem_edit->midavl;
one_mem edit->one mem show;

```

```

one_mem_edit->sex_code;
one_rent_add->titlebar;
one_rent_add->midavl;
one_rent_add->bidavl;
one_rent_edit->titlebar;
one_rent_edit->rmidavl;
one_rent_edit->one_rent_show;
one_rent_edit->bidavl;
one_ret_add->titlebar;
one_ret_add->rmidavl;
one_ret_add->one_rent_show;
one_ret_add->bidavl;
rent->titlebar;
rent->menu;
rent->rentadd;
rent->rentshow;
rent->rentfind;
rent->rentedit;
rentadd->one_rent_add;
rentedit->titlebar;
rentedit->one_rent_edit;
rentfind->titlebar;
rentfind->menu;
rentfind->by_rmid;
rentfind->by_rmname;
rentfind->by_rbid;
rentfind->by_rbname;
rentshow->titlebar;
rentshow->rnormal;
retadd->one_ret_add;
retshow->titlebar;
retshow->rnormal;
rnormal->titlebar;
rnormal->one_rent_show;
sex->sex_code;
sex->mnormal;
tnormal->titlebar;
tnormal->one_ret_show;
}

```



APPENDIX - B

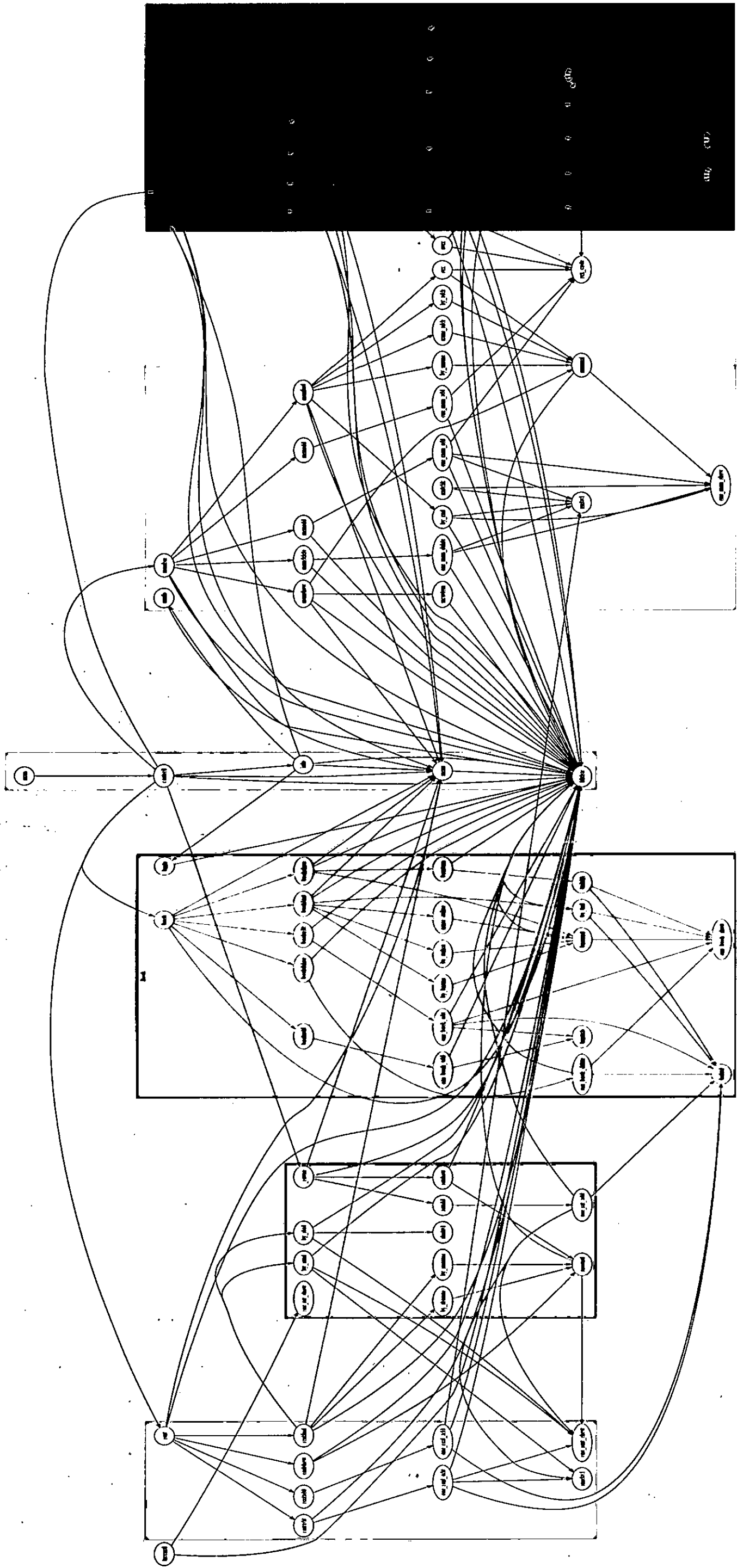


Fig. B.1 Gold Standard for Library Management System

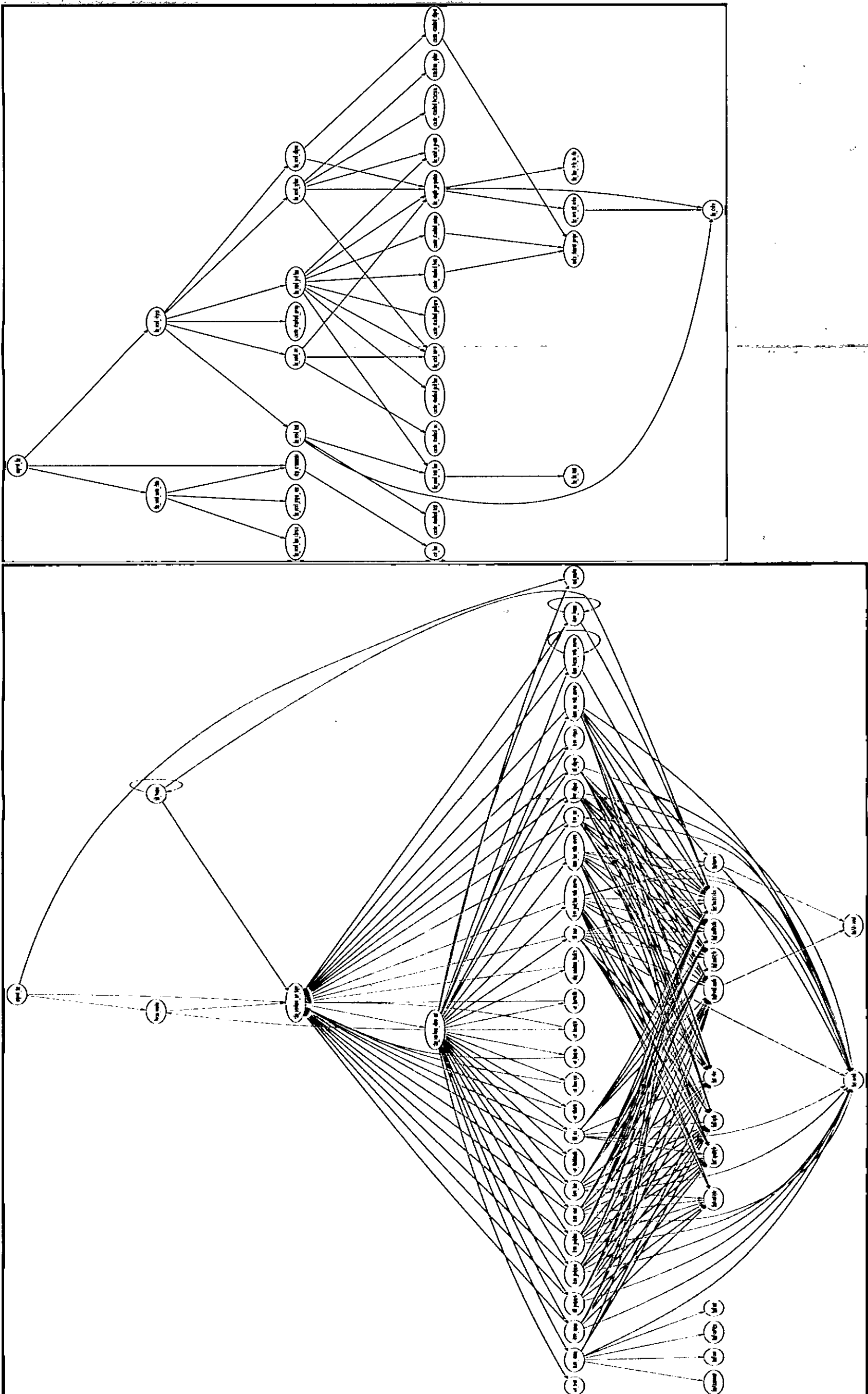


Fig. B.2 Gold Standard for Xfig

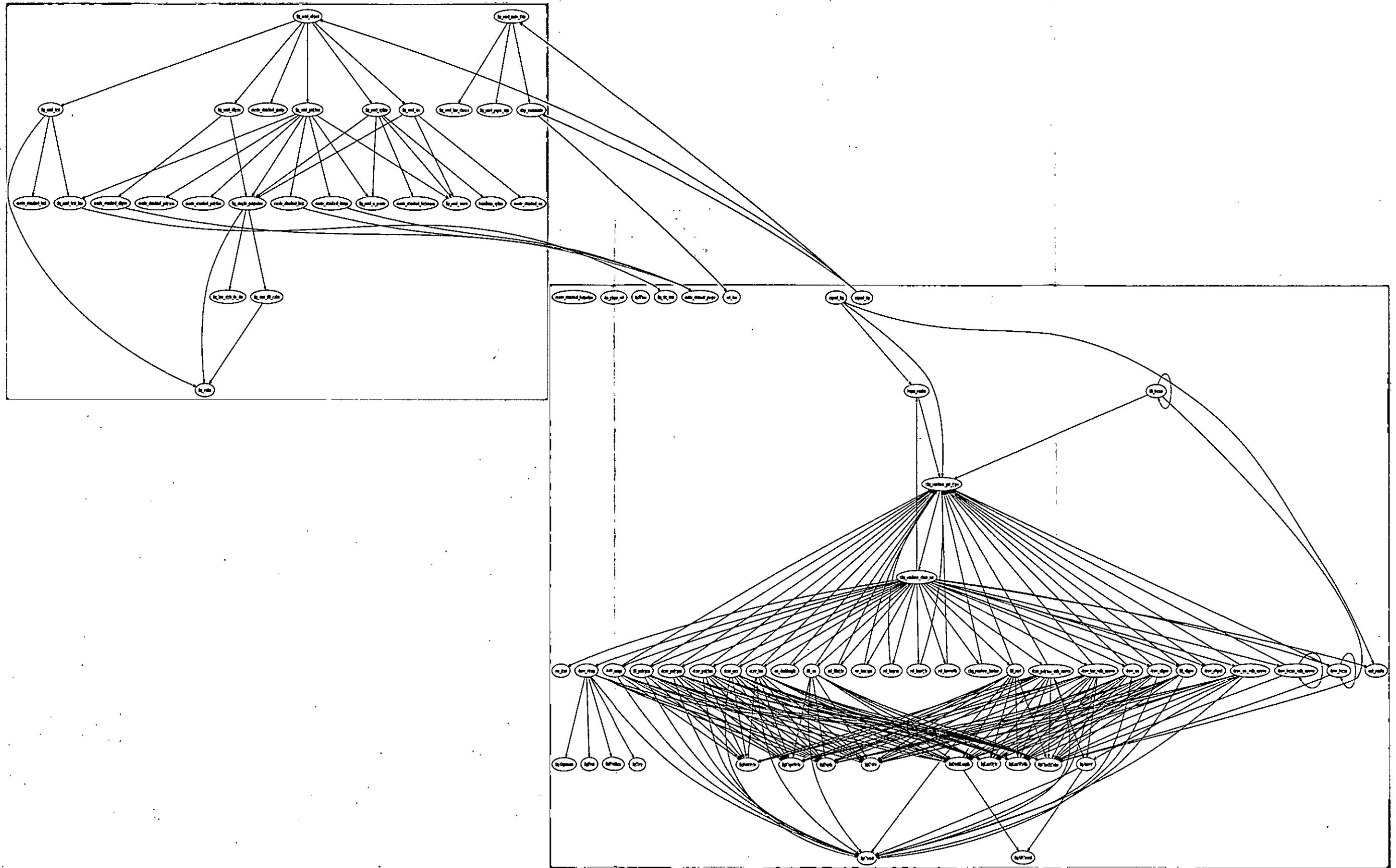


Fig. B.3 Clustering for Xfig from BUET Cluster 1.0





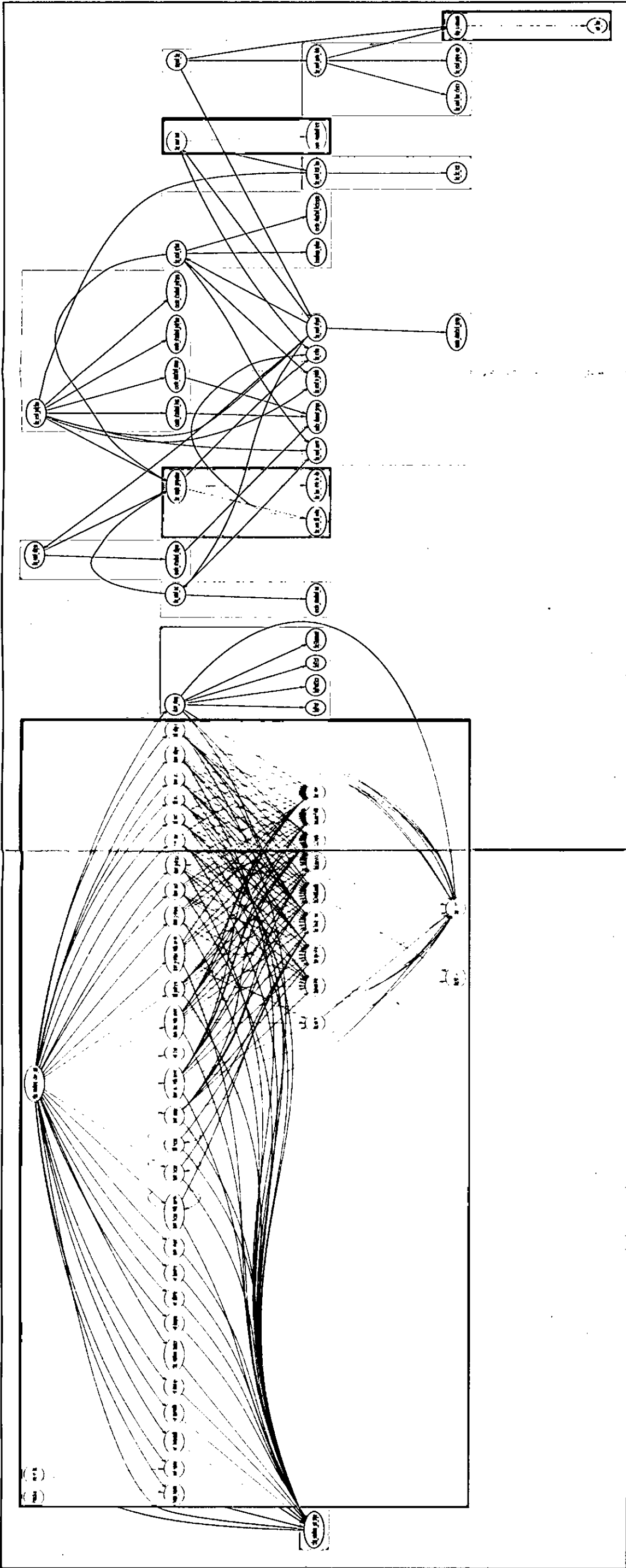


Fig. B.5 Clustering for Xfig from ACDC