ISSN: 2321-7782 (Online)
Volume 3, Issue 11, November 2015
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

Partitioning a Class-Oriented Model Graph (Software System) by Modified-MCL Algorithm

Appala Srinuvasu Muttipati[1]
Research Scholar
Dept. of Computer Science and Engineering
GITAM University
Visakhapatnam, India

Dr. Padmaja Poosapati[2]
Associate professor
Dept. of Information Technology
GITAM University
Visakhapatnam, India

Abstract: ***Object-oriented design is a paradigm to solve a problem that can be used in many different applications. For the design of a new application expansive measure of code can be reused by the existing design pattern. Hence, it is necessary to place identical design structures into a group. By identifying and analyzing these structures, we can obtain useful information about the design such as commonly used design patterns, design clones, most frequent design defects etc. This may help developers to improve knowledge about the software architecture. To form such clusters many clustering algorithms exist. One of the frequently used clustering algorithms is Markov Clustering (MCL) used in various applications like bioinformatics, biological networks, and community networks. MCL algorithm works well when the clusters are small, but when the clusters are large in diameter, there is a possibility that the weakly connected components may overlap producing overlapping clusters. In such cases, MCL algorithm cannot detect and eliminate the overlapping. To remove overlapping which will result in an inconsistent set of uni-class clusters, authors adopted and modified the existing Markov Clustering algorithm and thus proposed the new graph partitioning approach named as Modified-Markov Clustering Algorithm (Modified-MCL). The suggested algorithm interprets the clusters without overlapping by an iterative process of expansion and inflation operations. Tested with a hypothetical object-oriented system, results obtained by Modified-MCL were compared with the existing MCL algorithm and is observed that there is no overlapping between the clusters. The proposed Partitioning algorithm performs well for clustering object oriented systems.***

Keywords: object-oriented design; reusability; identical design structures; software clustering; graph partitioning

I. INTRODUCTION

Software systems are hard to comprehend the functionality of the large systems because systems are implemented with concepts of object oriented programming. Therefore, it becomes difficult to extend and modify software systems for the new application. Thus, the large and complex software systems can be divided into smaller components that are easier to understand by a software developer or designer.

The concept of breaking a large software system into small components can be achieved through clustering. Clustering is a useful and important unsupervised learning technique widely studied in the literature [1, 2, 3 and 4].The universal goal of clustering is to group similar objects into one cluster while partitioning dissimilar objects into different clusters. Clustering has broad applications including the analysis of biological data, financial data, time series data and spatial data so on.

Graph is an expressive data structure, which is widely used to model the structural relationship between objects in many application domains such as web, social networks, biological networks and fraud detection, etc. Graph clustering is an interesting and challenging research problem which has received much attention recently [5] Clustering on a large graph aims to partition the graph into several densely connected components. Typical applications of graph clustering include community

detection in social networks, identification of functionally related protein modules in large protein-protein interaction networks, etc [6].

Software clustering is an important feature to divide a software system into smaller subsystems. It helps to discover the software subsystems that are interrelated functionally and independent from each other part of the system. Therefore, the reengineering processes need to identify the subsystem of an activity to concern with the regeneration and improvement of existing software application.

Designing reusable software (i.e. modular) system is an important principle in Object-oriented domain. A modular software system is a single unit of the application (software) design. Each module is encapsulated with a small number of classes that are strongly coupled. Modules must have a simple interface through which they can interrelate with each other modules.

The rest of the paper is organized as follows. Section 2 describes the related work. In section 3, proposed approach Modified-MCL algorithm is presented. Section 4 describes the case study. Section 5 concludes our work.

II. Related Work

Several techniques and approaches for software clustering are introduced in the study. Shtern and Tzerpos [7] presented a review article on clustering methodologies for software engineering. Authors described each phase of a clustering algorithms separately and important approaches for evaluating the effectiveness of software clustering

Spectral graph partitioning methods first appeared in the early seventies in research work of Donath and Hoffman [8], Fiedler [9][10]. They explored properties of algebraic representations of the graphs and introduced the idea of using eigenvectors for partitioning the graph. These methods have been applied to many research disciplines, related to computer science and electrical engineering. Hendrickson and Leland [11] used Spectral Graph Partitioning in parallel systems. Pothen et. al. [12] and Bernard and Simon [13] used similar techniques in scientific computing.

Xanthos [14] proposed a spectral graph partitioning method to decompose the object-oriented software system. The methodology is based on an iterative process for partitioning the graph in order to identify dense communities of classes thereby minimizing the communication between modules of the system. This is achieved because it finds the minimum cut set of edges in each iteration. Therefore, it can also be utilized to identify the modules that should be assigned in diverse machines, in a distributed environment.

Hierarchical clustering algorithms are two categories: agglomerative is a bottom-up approach and divisive is a top-down approach. The agglomerative algorithm starts from the bottom of the hierarchy by iteratively grouping similar entities into clusters. At each step, the two clusters that are most similar to each other are merged, and the number of clusters is reduced by one. Divisive algorithms start with one cluster that contains all entities and divide the cluster into a number (usually two) of separate clusters at each successive step.

Czibula and Serban [15] presented a new hierarchical agglomerative clustering algorithm utilized for object-oriented software systems restructuring. This methodology aims at identifying a partition of a software system that corresponds to an improved structure of it. Hierarchical Agglomerative clustering algorithm for Restructuring Software systems (HARS) can be used in the grouping step of Clustering Approach for Refactoring Determination (CARD) in order to re-group entities from the software system. This methodology can be useful for assisting software engineers in their daily works of refactoring software systems. They evaluated using the open source JHotDraw (i.e. a java GUI framework for technical and structured graphics) emphasizing its advantages in comparison with existing approaches.

Erdemir et. al., [16] proposed an object-oriented software clustering approach based on the adaption of the fast community detection algorithm. It is an agglomerative hierarchical clustering algorithm specifically designed for finding communities in social network graphs within the small-world behavior. The algorithm was implemented on directed weighted graph. Authors

evaluated the algorithm and compared with previously designed approaches. It is observed that the algorithm has considerable speed average over other algorithms.

Han and Kamber [17] described normalization which is one of the data transformation methods in data pre-processing. An attribute of a dataset is normalized by scaling its value so that they fall within a small range, such as 0.0 to 1.0. Normalization may improve the accuracy and efficiency of mining algorithms in classification and clustering. There are numerous methods for data normalization that include min-max normalization, z-score normalization and normalization by decimal scaling.

Dongen [18] suggested a Markov Cluster (MCL) Algorithm, which involves changing the values of a transition matrix towards either 0 or 1 at every progression in a random walk until the stochastic conditions are satisfied. When the Hadamard power for each transition probability value is divided by the sum of each column, the rescaling procedure yields a transition matrix for the next stage. After repeating alternatively for around 20 times between two stages random walk and probability modification, the procedure will at last achieve a convergence stage in which the whole graph is subdivided into a set of 'hard' clusters.

Yu, Ge and Wu [19] proposed a kind of weighted graph to represent the system and design pattern. The nodes in the graph represent the classes, whereas edges are represented as the association between two corresponding classes. For indicating different relationships between classes, they provide each kind of relationship as unique prime number as its weight. The relationship weights and relationships are '2' correspond to association, '3' corresponds to inheritance, '5' corresponds to aggregation, '6' corresponds to the combination of association and inheritance, '10' corresponds to the combination of association and aggregation, '15' corresponds to the combination of inheritance and aggregation and '30' corresponds to the combination of association , inheritance and aggregation.

Muttipati and Padmaja [20] describe a labelled graph to represent the software system. The vertices of the generated graph are classes, abstract and interface. The edges are the associations between entities. Association types in the graph are based on associations of UML class diagrams. Normally there can be more than one association between entities (classes, abstracts, interfaces). Therefore to create a simple graph all the parallel edges are merged into one single edge. The feasible association types between classes are (i) Extend association X, where Class B is the base class of Class A, (ii) Implement association I, where Class A implements the interface of class B, (iii) Field type association A, where Class A has a field type of Class B, (iv) Local variable association L, where Class A method has a local parameter with the type of Class B, (v) Parameter association P, where Class A method has an input parameter with the type of Class B, (vi) Return type association R, where Class A has a method with the return type of Class B, (vii) Method association M, where Class A has a method calls to Class B. (viii)override association O, where Class A has an overrides method of Class B. Fig 1 describes the Class-Oriented Model Graph generation based on UML and AST.

III. Graph Representation of Class-Oriented Model Graph

This session explains the constructed class-oriented model graph. We represent the high-level view of software architecture as a weighted graph. In UML Class diagram are directly edges weights are considered based on discrete messages exchanged between the classes. The vertices/nodes of this graph are classes, abstract class, and interfaces. The edges of the graph represent the links between the entities. Connecting lines between classes in the diagram are called links. These associated instances are links. This flow indicates the message sending between two classes where weight on edges represents the total communications between the classes. Figure 1 indicated system flow.

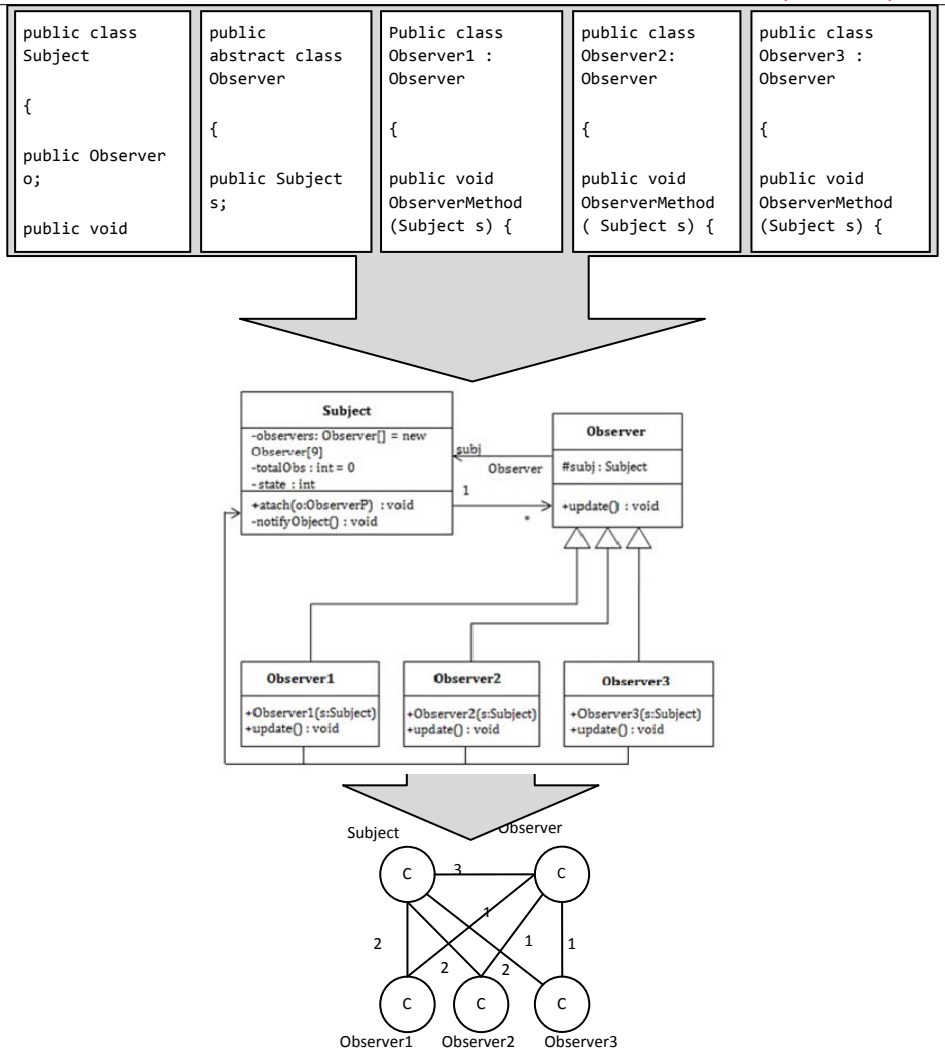

Fig. 1 Flow diagram for the partitioning of Class-oriented model graph

Definition Class-Oriented Model Graph (G) is a weighted graph representing a set of classes and their association between them, which is defined as 3- tuple G = (V,E, v), where

» V is a non empty set of vertices which represent classes.

» E ⊂ V×V is a non-empty set of edges which represent associations between classes

» V: E→WE is the function assigning weights to the edges.

IV. PROPOSED APPROACH

The proposed approach is partitioning a software system into subsystems component. The software clusters are generated from the class-oriented model graph. Authors have suggested a modified graph partitioning algorithm by adopting few features of MCL algorithm. In this algorithm symmetric matrix M is formed by combining associate matrix and degree matrix of G. The column normalization technique is applied to matrix M for transforming the data to a specified small range. The algorithm consists of two main operations namely inflation and expand which are implemented on the normalized matrix. These operations are repeated alternatively until stable steady matrix is reached. Finally the clusters are interpreted from stable matrix S (i, j), where $1 \leq i \leq n$, $1 \leq j \leq n$, and n is number of vertices. For the rows having the value one, the corresponding j indices are taken into one group called cluster. Hence in this way clusters are interpreted from stable matrix. Figure 1shows the block diagram of the proposed approach modified- MCL.

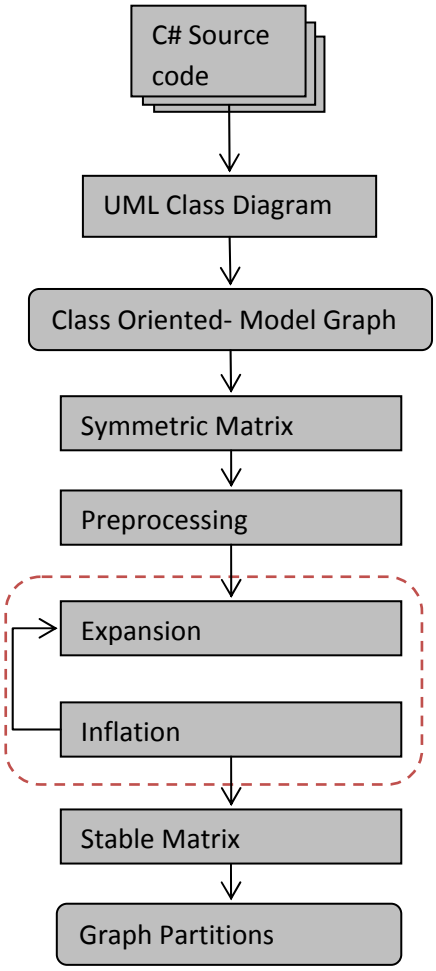

Fig. 2 Block Diagram for Proposed Approach

a) Generating software clusters by Modified-MCL algorithm

The input for the modified-MCL algorithm is an undirected weighted graph G, inflation parameter 'm' and extand parameter 'b'. From the input graph G, an associate matrix A is created and is represented as A= $[a_{ij}]$, where a_{ij} is the weight of the edge e_{ij}. Degree matrix D of G is found, which is represented as D= $[d_{ij}]$ and is defined as

$$d_{ij} = \begin{cases} \sum_{k=1}^{n} a_{ik} & if\ i = j \\ 0 & otherwise \end{cases} \quad (1)$$

Associate and Degree matrix of G are combined to obtain the symmetric matrix M.

The resultant symmetric matrix M is normalized by minimum and the maximum value of the column of a matrix M is taken and the new range for the matrix is defined from [0, 1]. Here, new_min_A= 0and new_max_B=1.Thus the above equation for the matrix M (i, j) can be written as

$$v' = \frac{X}{Y} * (new\_max\_B - new\_min\_A) + \ new\_min\_A \quad (2)$$

Where X = [v-min value of jth column], Y= [max value of jth column – min value of jth column] and $1 \leq j \leq n$, n is number of vertices

After normalizing the matrix (M_N) two main operations, namely expansion and inflation are applied. These operations are repeated iteratively until a convergence is reached i.e. until a stable matrix is reached.

The expansion matrix M_E is achieved via matrix squaring or matrix multiplication and is defined as

$$M_E=\sum_{k=1}^{m} M_{ik}M_{kj} \quad (3)$$

Where, m is an integer taken as 2.

Clusters of large size are formed with the higher value of m. This may lead to a single cluster with no partitions and lower value of m (i.e. m=1) may lead to the problem of multiplication. Therefore m value is considered as 2.

The inflation is achieved by element square of the expansion matrix M' and element squared matrix is normalized. The expansion operation on the present state is responsible for both strengthening due to element squaring and weakening due to normalizing. Therefore to control the extent of the strengthening / weakening, inflation parameter b is considered. The low inflation parameter is more prone to yield smooth partitions.

$$M_I=M''_{ij} = \frac{(M'_{ij})^b}{\text{column sum}} \quad (4)$$

Where, b is an integer value taken as 2.

The expansion and inflation operations are iteratively repeated until it results in a stable state matrix.

Finally the clusters are interpreted from stable matrix S (i, j), where $1\leq i \leq n$ and $1\leq j \leq n$, n is a number of vertices. For the rows having the value one, the corresponding j indices are taken into one group called a cluster. Thus in this way the clusters are interpreted from the stable matrix.

Algorithm: Modified-Markov Clustering Algorithm (Modified-MCL)

Input: Class-Oriented Model Graph G, expansion parameter '*m*', inflation parameter '*b*'

Output: Partitions of Class-Oriented Model Graph

Step 1: Create an associate matrix A of G

Step 2: Find the degree of the matrix D of G

Step 3: Combine the associate matrix and degree matrix to form a symmetric matrix M i.e. M = A+D

Step 4: The symmetric Matrix M is normalized M_N using min-max normalization technique.

Step 5: Apply simple matrix multiplication $M_N \times M_N$ to obtain expansion matrix M_E.

Step 6: Apply element squaring for obtain expansion matrix and column normalization for the element squared matrix to obtain inflation matrix M_I.

Step 7: Repeat step 4 and step 5 until a stable state matrix M_S is obtained.

Step 8: Finding the number of ones in a row and combine all the indices into one group to form a cluster from stable matrix generated in step 7.

V. CASE STUDY

Let us consider a hypothetical object-oriented system consisting of XML Generation, a GUI part and Database part, as shown in figure3.

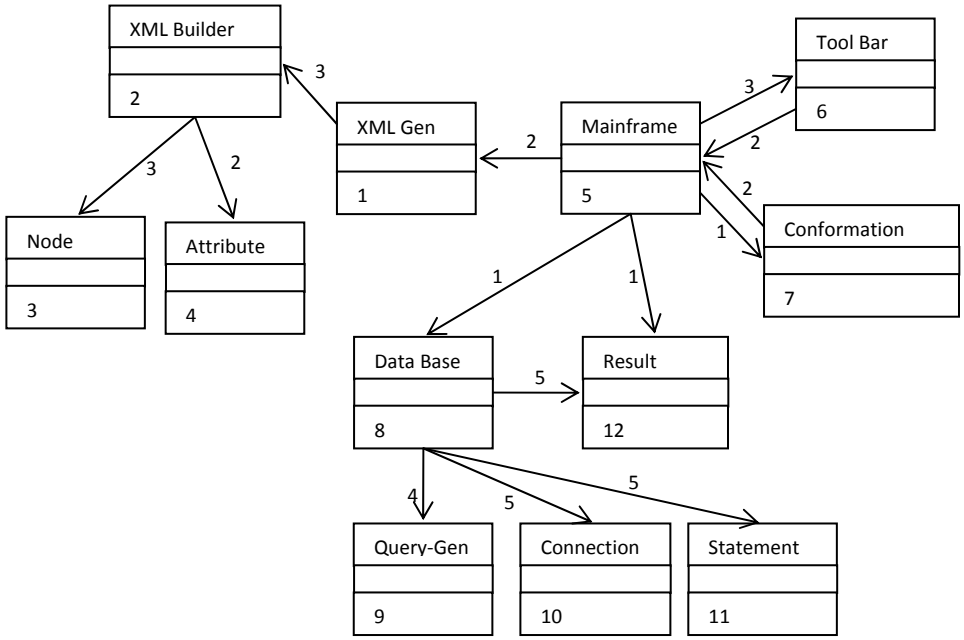

Fig.3 software design with three discrete parts

The corresponding undirected graph results if classes are mapped to vertices and edges between them are weighted according to the total number of messages exchanged in both directions. The graph is shown in fig.4.

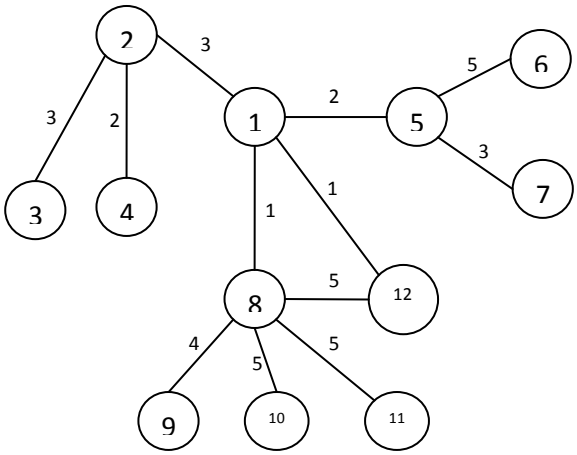

Fig.4 corresponding undirected graph

The association matrix, degree matrix and symmetric matrix for corresponding undirected graph are shown below:

$$A = \begin{bmatrix} 0 & 3 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 3 & 0 & 3 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 5 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 5 & 5 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = (A + D) = \begin{bmatrix} 4 & 3 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 3 & 3 & 3 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 3 & 5 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 4 & 5 & 5 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The obtained symmetric matrix is pre-processed by Min-Max normalization technique within the new range between [0, 1]. From the matrix M, first column elements are [4, 3, 0, 0, 2, 0, 0, 1, 0, 0, 0, 1], where the maximum value is 4 and the minimum value is 0. The value of M (1, 1) i.e. 4 is normalized to ((4-0) / (4-0)) *(1-0) + 0 =1 and value of M (2, 1) i.e. 2 is normalized to ((3-0) / (4-0)) * (1-0) + 0=0.75

Similarly Computing for entire matrix, normalized matrix M_N is

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0.4 & 0 & 0 & 0.2 & 0 & 0 & 0 & 0.2 \\ 0.75 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0.33 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.67 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0.6 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.6 & 0 & 0.33 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0.25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0.2 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0.2 \end{bmatrix}$$

On normalized matrix, expansion and inflation operations are applied. The expansion operation is a simple matrix squaring (i.e. $M_N \times M_N = (M_N)^2$). Thus the expansion matrix M_E is

$$\begin{bmatrix} 2.05 & 2 & 1 & 1 & 0.64 & 0.4 & 0.4 & 0.6 & 0.2 & 0.2 & 0.2 & 0.44 \\ 1.5 & 3.416 & 1.333 & 1.5 & 0.3 & 0 & 0 & 0.15 & 0 & 0 & 0 & 0.15 \\ 0.75 & 1.333 & 1.111 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 1 & 0.667 & 0.916 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.8 & 0.5 & 0 & 0 & 2.16 & 0.8 & 0.933 & 0.1 & 0 & 0 & 0 & 0.1 \\ 0.5 & 0 & 0 & 0 & 0.8 & 1.04 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0.3 & 0 & 0 & 0 & 0.56 & 0.6 & 0.711 & 0 & 0 & 0 & 0 & 0 \\ 0.75 & 0.25 & 0 & 0 & 0.1 & 0 & 0 & 4.85 & 1.25 & 1.2 & 1.2 & 1.25 \\ 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.862 & 0.8 & 0.8 & 0.8 \\ 0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 1.2 & 1 & 1.04 & 1 & 1 \\ 0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 1.2 & 1 & 1 & 1.04 & 1 \\ 0.55 & 0.25 & 0 & 0 & 0.1 & 0 & 0 & 1.25 & 1 & 1 & 1 & 1.09 \end{bmatrix}$$

The inflation operation is combination of matrix squaring and vertex normalization. First, the matrix element squaring, illustrating by taking the square of the first column element of M_E, that results with the values (2.05, 1.5, 0.75, 0.5, 0.8, 0.5, 0.3, 0.75, 0.2, 0.25,0.25, 0.55). Squaring is done for the remaining columns of matrix M_E resulting in a matrix as shown below.

$$\begin{bmatrix} 4.202 & 4 & 1 & 1 & 0.409 & 0.16 & 0.16 & 0.36 & 0.04 & 0.04 & 0.04 & 0.194 \\ 2.25 & 11.673 & 1.777 & 2.25 & 0.09 & 0 & 0 & 0.023 & 0 & 0 & 0 & 0.023 \\ 0.562 & 1.777 & 1.234 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.25 & 1 & 0.444 & 0.840 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.64 & 0.25 & 0 & 0 & 4.666 & 0.64 & 0.871 & 0.01 & 0 & 0 & 0 & 0.01 \\ 0.25 & 0 & 0 & 0 & 0.64 & 1.082 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0.09 & 0 & 0 & 0 & 0.313 & 0.36 & 0.505 & 0 & 0 & 0 & 0 & 0 \\ 0.562 & 0.0625 & 0 & 0 & 0.01 & 0 & 0 & 23.523 & 1.5625 & 1.44 & 1.44 & 1.563 \\ 0.04 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.7439 & 0.64 & 0.64 & 0.64 \\ 0.063 & 0 & 0 & 0 & 0 & 0 & 0 & 1.44 & 1 & 1.082 & 1 & 1 \\ 0.063 & 0 & 0 & 0 & 0 & 0 & 0 & 1.44 & 1 & 1 & 1.082 & 1 \\ 0.303 & 0.063 & 0 & 0 & 0.01 & 0 & 0 & 1.563 & 1 & 1 & 1 & 1.188 \end{bmatrix}$$

Second, for the obtained result compute the vertex normalization method (i.e. M (i, j)/ column sum) as follows: from the above matrix first element is 4.202 and sum of the first column is 9.275. 4.202/9.275 = 0.453. Thus the resulting inflation Matrix M_I is

$$\begin{bmatrix}
0.453 & 0.212 & 0.224 & 0.196 & 0.067 & 0.071 & 0.063 & 0.012 & 0.007 & 0.007 & 0.007 & 0.034 \\
0.243 & 0.620 & 0.399 & 0.442 & 0.014 & 0 & 0 & 0.0007 & 0 & 0 & 0 & 0.004 \\
0.060 & 0.094 & 0.277 & 0.196 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.027 & 0.053 & 0.099 & 0.165 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.069 & 0.013 & 0 & 0 & 0.760 & 0.285 & 0.343 & 0.0003 & 0 & 0 & 0 & 0.001 \\
0.027 & 0 & 0 & 0 & 0.104 & 0.482 & 0.394 & 0 & 0 & 0 & 0 & 0 \\
0.009 & 0 & 0 & 0 & 0.051 & 0.160 & 0.199 & 0 & 0 & 0 & 0 & 0 \\
0.060 & 0.003 & 0 & 0 & 0.001 & 0 & 0 & 0.801 & 0.292 & 0.276 & 0.276 & 0278 \\
0.004 & 0 & 0 & 0 & 0 & 0 & 0 & 0.034 & 1.139 & 0.123 & 0.123 & 0.113 \\
0.006 & 0 & 0 & 0 & 0 & 0 & 0 & 0.049 & 0.187 & 1.27 & 0.192 & 0.178 \\
0.006 & 0 & 0 & 0 & 0 & 0 & 0 & 0.049 & 0.187 & 0.192 & 1.207 & 0.178 \\
0.032 & 0.003 & 0 & 0 & 0.001 & 0 & 0 & 0.053 & 0.187 & 0.192 & 0.192 & 1.211
\end{bmatrix}$$

The expansion and inflation operations are repeated alternatively on matrix M_I until the matrix gets a stable state. Finally a stable matrix M_S is obtained as

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

From the matrix M_S, the partitions are interpreted by finding the number of ones in a row and combining the corresponding indices into one group. Vertices 1, 2, 3, and 4 are considered as one group or one partition, since all the ones in the first row of M_S correspond to the indices 1, 2, 3, 4. Similarly vertices 5, 6 and 7 belong to another group forming partition 2, vertices 8, 9, 10, 11, and 12 belongs to another group forming partition 3.

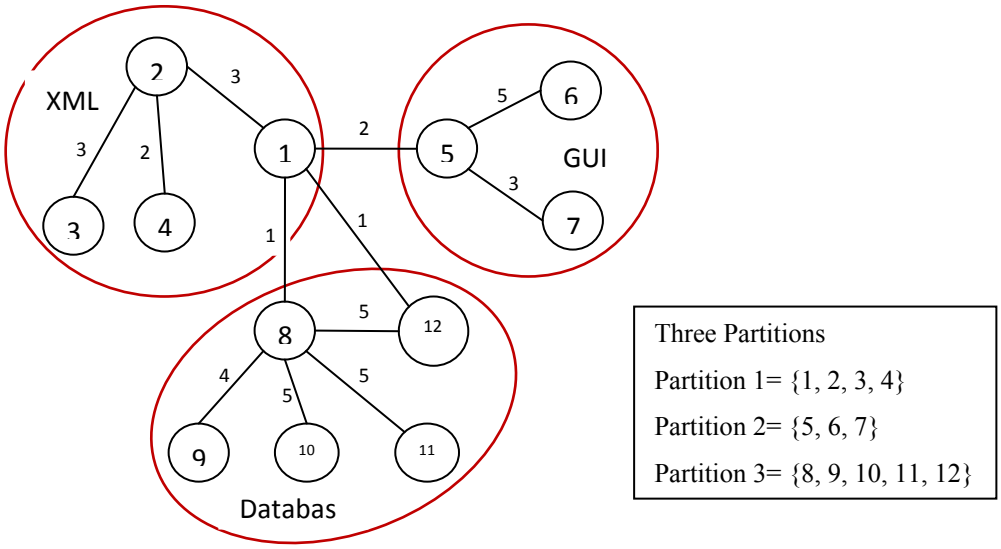

Fig. 5 Class-Oriented Model Graph Partitions with Modified-MCL Algorithm

We observed the results of Modified-MCL and MCL algorithms on Hypothetical Object-Oriented system. In both algorithms three partitions are observed but in the MCL algorithm overlapping is occurring with the vertex 1. The comparison of the Modified-MCL and MCL algorithms are shown in Table I.

Table I. Comparison of Modified-MCL and MCL algorithm on hypothetical object-oriented system

algorithm	Vertices	Edges	Total partitions	Partition 1 elements	Partition 2 elements	Partition 3 elements	Overlapping
Modified-MCL	12	12	3	{1, 2, 3, 4}	{5, 6, 7}	{8, 9, 10, 11, 12}	No
MCL	12	12	3	{1, 2, 3, 4}	{1,5, 6, 7}	{1, 8, 9, 10, 11, 12}	Yes

VI. Conclusion

In this paper, a graph partitioning approach is discussed by developing a Modified-MCL algorithm and to provide non-overlapping partitions in object oriented system. This can be achieved by pre-processing two main operations, one is simple matrix multiplication which is called expansion and another one is squaring the elements of the expansion matrix and column normalizing the element squared matrix which is called inflation. These operations are repeated alternatively to yield a stable matrix state to form clusters. The suggested Modified-MCL algorithm was examined on a hypothetical object-oriented system. The stable matrix is occurring without overlapping of vertices in partitions when compared with MCL algorithm as it is having an overlap of vertices in partitions. Authors would like to come up with real-time applications and to find an optimal solution for object-oriented systems.

References

1. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, “Automatic subspace clustering of high dimensional data for data mining applications”, in Proc. ACM-SIGMOD’98, 1998, pp.94-105.
2. M. Ester, H. P. Kriegel, J. Sander and X. Xu, “Density-Based Clustering in Spatial Databases: The algorithm GDBSCAN and Its Applications,” Data Mining and Knowledge Discovery, vol. 2, pp.169-194, 1998.
3. D. Gibson, J. Kleinberg and P. Raghavan, “Inferring web communities from link topology,” in Proc, HYPERTEXT’98, 1998, pp.225–234.
4. R. T. Ng and J. Han, “Efficient and effective clustering method for spatial data mining,” in Proc. VLBD’94, 1994, pp.144–155.
5. D. A. Bader, H. Meyerhenke, P. Sanders and D. Wagner, D (2012) ‘Graph Partitioning and Graph Clustering, 10th DIMACS Implementation Challenge Workshop, February 13-14, 2012, Georgia Institute of Technology, Atlanta, GA. Contemporary Mathematics 588, American Mathematical Society and Center for Discrete Mathematics and Theoretical Computer Science, 2013.
6. S. E. Schaeffer, “Graph Clustering,” Computer Science Review, pp.27-64, 2007.
7. M. Shtern and V. Tzerpos, “Clustering methodologies for software engineering,” Advances in Software Engineering, Vol. 2012, Article ID 792024, 18 pages, 2012.
8. W. E. Donath and A. J. Hoffman, “Lower bounds for the partitioning of graphs,” IBM Journal of Research and Development, vol. 17, pp.420-425, 1973.
9. M. Fiedler, “Algebraic connectivity of graphs,” Czechoslovak Mathematical Journal, vol. 23, pp.298-305, 1973.
10. M. Fiedler, “A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory,” Czechoslovak Mathematical Journal, vol. 25, pp.619-633, 1975.
11. B. Hendrickson and R. Leland, “An improved spectral graph partitioning algorithm for mapping parallel computations,” SIAM Journal on Scientific Computing, vol. 16, pp.452-469, 1995.
12. A. Pothen, H. D. Simon and K. P. P. Liu, “Partitioning sparse matrices with eigenvectors of graphs source,” SIAM Journal on Matrix Analysis and Applications, vol. 11, pp.1-30, 1990.
13. S. Barnard and H. Simon, (1994) “A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems,” Concurrency: Practice and Experience, vol. 6, pp.101-117, 1994.
14. S. Xanthos, ‘Clustering Object-Oriented Software Systems using Spectral Graph Partitioning’ In: ACM Student Research Competition, Grand Finals, Second Award, 2005.
15. I. G. Czibula and G. Serban, “Hierarchical Clustering for Software Restructuring,” Babes Bolyai University, Romania, 2007.
16. U. Erdemir, U. Tekin and F, Buzluca, “Object oriented software clustering based on community structure,” in Proc. APSEC’11, 2011, pp.315-321.
17. J. Han and M. Kamber, Data Preprocessing, Data Mining Concepts and Techniques, Morgan Kaufmann Publishers, an imprint of Elsevier, pp.105-140. 2006.
18. S.V. Dongen, Graph clustering by flow simulation. PhD thesis, University of Utrecht, Netherland, 2000.
19. Y. Dongjin, J. Ge; W. Wu, "Detection of design pattern instances based on graph isomorphism," in Proc. ICSESS'13, 2013, pp.874-877
20. A. S. Muttipati and P. Padmaja “Construction of Software Model Graph and Analyzing Object-Oriented Program (C#) Using Abstract Syntax Tree Method” International Journal of Computer Science and Information Technologies, vol.6 pp, 2015.

AUTHOR(S) PROFILE

Appala Srinuvasu Muttipati is a Research Scholar in Computer Science and Engineering from GITAM University, Visakhapatnam, Andhra Pradesh, India. He received his M.Tech Degree in Computer Science and Technology from GITAM University in 2011. MCA Degree from Andhra University in 2008. His Current research areas include Data mining and Graph mining.

Padmaja Poosapati is an Associate Professor in Department of Information Technology at GITAM University, Visakhapatnam, Andhra Pradesh, India. She received her Master degree in Computer Science and Engineering from Andhra University in 1999 and Ph.D. degree in Computer Science and Engineering from Andhra University 2010. Her current research interests include Clustering and Classification in Data mining and Graph mining.