12-2011

Application Oriented Analysis of Large Scale Datasets

Prashant Shivaji Paymal
University of Nebraska at Omaha

Application Oriented Analysis of Large Scale Datasets

A Thesis

Presented to the

Department of Computer Science

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Major: Computer Science

University of Nebraska at Omaha

by

Prashant Shivaji Paymal

December, 2011

<u>Supervisory Committee:</u>

Dr. Sanjukta Bhowmick

Dr. Hesham Ali

Dr. Harvey Siy

Dr. Dhundy Bastola

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Application Oriented Analysis of Large Scale Datasets

Prashant Shivaji Paymal, M.S.

University of Nebraska, 2011

Advisor: Dr. Sanjukta Bhowmick

Abstract

Diverse application areas, such as social network, epidemiology, and software engineering consist of systems of objects and their relationships. Such systems are generally modeled as graphs. Graphs consist of vertices that represent the objects, and edges that represent the relationships between them. These systems are data intensive and it is important to correctly analyze the data to obtain meaningful information. Combinatorial metrics can provide useful insights for analyzing these systems. In this thesis, we use the graph based metrics such as betweenness centrality, clustering coefficient, articulation points, etc. for analyzing instances of large change in evolving networks (Software Engineering), and identifying points of similarity (Gene Expression Data). Computations of combinatorial properties are expensive and most real world networks are not static. As the network evolves these properties have to be recomputed. In the last part of thesis, we develop a fast algorithm that avoids redundant re-computations of communities in dynamic networks.

Acknowledgement

I am extremely grateful and indebted to my thesis advisor, Dr. Sanjukta Bhowmick, for accepting me as her student. Her constant guidance, encouragement, and support throughout the completion of my work were the key for my thesis. I owe a special thanks to Dr. Hesham Ali and Dr. Harvey Siy for their invaluable guidance and enlightening discussions on the topic which made this thesis possible. I am also grateful to Dr. Dhundy Bastola for providing valuable feedback and comments on my thesis.

I would like to thank all the professors of the Computer Science department at University of Nebraska - Omaha, who taught great courses. I would also like to thank several students in our research group, and my friends for their support, motivation, and help during my stay here at Omaha.

Most importantly, I would like to thank my mother and father, Kanchan and Shivaji Paymal, and brother Kedar, who have cared for me continuously, in every possible way, from halfway around the world.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Analysis of large datasets is a crucial component in advancing our understanding in diverse applications areas, such as social networks [1], epidemiology [2] and software engineering. The data from these fields are generally represented as systems of interacting entities. Two popular methods of expressing this information are (i) as networks where vertices are the objects and edges associated relations (for example, social networks) or (ii) as matrices where the rows represent the entities and the columns the features defining them (for example differentially expressed levels of genes). Most analysis techniques are application agnostic that is they are not designed with the end objective in mind. The mathematical models are rarely corroborated from an application user's point of view. In this thesis, we demonstrate how combinatorial properties relate to application characteristics and validate our results by analyzing evolving networks from two very different application areas; software engineering and bioinformatics.

Understanding how networks evolve over time is an important analysis task. However, due to the large number of components in most real world systems, it is difficult to get a quick summary of network evolution. Therefore, there has been little study in understanding the change in dynamic networks. In the first part of this thesis, we explore combinatorial metrics to quantify the difference between networks representing the evolution of JHotDraw software over several versions.

In the second part of this thesis, we explore combinatorial metrics to find the similarities between networks. We apply our similarity criteria to develop a new biclustering algorithm for improve analysis of microarray data. Biclustering represents an ideal approach for mining meaningful relationships from the massive data because it allows simultaneous clustering of both the entities and conditions.

Computation of combinatorial properties is a key to network analysis. However, real world networks are not static they evolve with the time. Therefore, for each evolution the graph properties have to be recomputed. In the final part of this thesis, we develop community detection algorithm, an important network characteristics, that reduces redundant computations on dynamic networks.

1.1 Contribution

Given below is list of our significant contributions,

- We have explored combinatorial metrics to quantify and evaluate the difference between networks. Our results provide important insights in understanding the rate of evolution networks.
- We have done a comprehensive research on different biclustering algorithms and developed a new biclustering algorithm based on network similarity.
- We have designed an efficient community detection algorithm for real-time dynamic networks that takes advantage of the information computed in previous time steps to avoid extra computations.

1.2 Outline of Thesis

This thesis is organized as follows. In Chapter 2, we discuss background information about graph theory. In Chapter 3, we present use of combinatorial metrics to analyze the evolution of networks representing JHotDraw software. In Chapter 4, we explore the combinatorial properties to find the similarities between networks and present a new biclustering algorithm for analysis of microarray data. In Chapter 5, we study analysis of dynamic networks and present community detection algorithm for dynamic networks. In Chapter 6, we discuss our concluding remarks and present potential ideas for further research.

Chapter 2

Background

A graph is a mathematical object that captures the notion of connection. Many problems of practical interest can be represented by graphs. In computer science, graphs are used to represent different networks such as social networks, software engineering networks, and biological networks, etc. Each of these networks consists of set of vertices and edges. For instance, people in the social networks, classes in the software engineering networks represent vertices in a graph and connection between people and classes represent edges in the social networks and software engineering networks respectively. Here, we introduce some network or graph terminology (based on the definitions provided in [3]). We classify the list of graph properties as, (i) vertex based properties, and (ii) network based properties. Vertex based properties are defined per vertex of the network and network based properties are defined over entire network.

2.1 Graph Terminology

A graph is collection of vertices and edges. Formally, $G = (V, E)$ consists of set of vertices V and edges E, where $E \subseteq V \times V$. There are two types of graphs directed and undirected. A graph is directed if edges point in one direction from one vertex to another vertex, otherwise a graph is undirected. A directed graph $G = (V, E)$ consists of a finite, nonempty set of vertices V and a set of edges E. Each edge is an ordered pair (v, w) of vertices. An undirected graph $G = (V, E)$ consists of a finite, nonempty set of vertices V and a set of edges E. Each edge is a set $\{v, w\}$ of vertices.

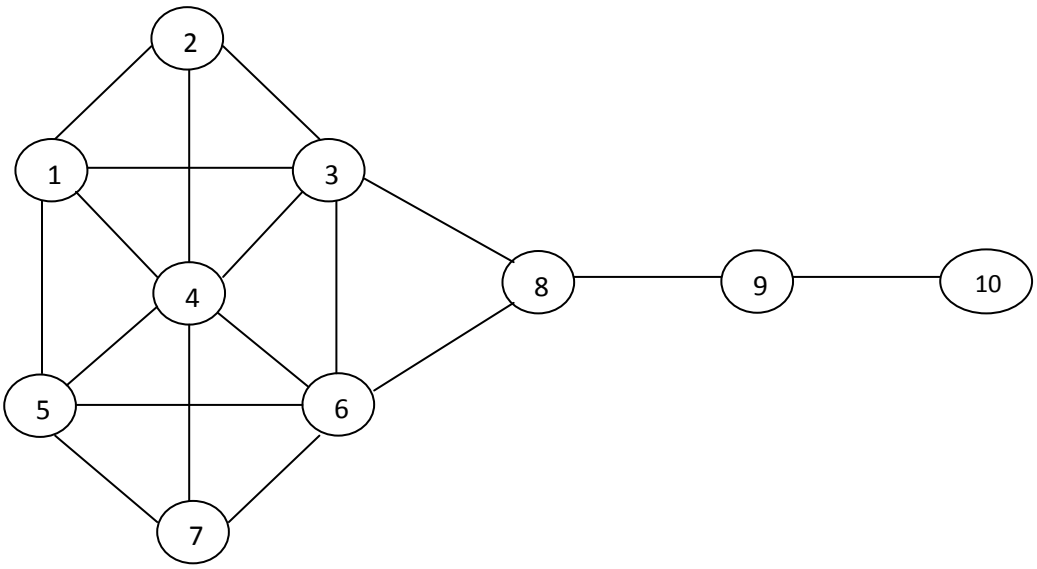


Figure 2.1: *Undirected Graph*

Graph Properties

2.1.1 Vertex Based Properties

- **Degree**

The degree of a vertex in a graph is the number of edges the vertex has with the other vertices. The degree of vertex v is denoted as $\deg(v)$. In directed graph, vertices have two different degrees, in-degree: the number of incoming edges and out-degree: the number of outgoing edges. In Figure 2.1, degree of vertices are, deg (V1) = 4, deg (V2) = 3, deg (V3) = 5, deg (V4) = 6, deg (V5) = 4, deg (V6) = 5, deg (V7) = 3, deg (V8) = 3, deg (V9) = 2, deg (V10) = 1.

- **Betweenness Centrality**

Most of the shortest paths in a network go through the vertices with the high betweenness centrality. Therefore, these vertices become more the central point controlling the communication. Betweenness Centrality of a vertex v is calculated as

sum of the ratio of the number of shortest path in the graph include vertex v to the total number of shortest path in the graph. The betweenness centrality $BC\ (v)$ of a vertex $v\ \in V$ is the sum over all pairs of vertices $u, w\ \in V$, of the fraction of shortest paths between u and w that pass through v

$$BC\ (v) = \sum_{\substack{u,w \in V \\ u \neq w \neq v}} \frac{\sigma_{u\,w}(v)}{\sigma_{u\,w}}$$

Where $\sigma_{u\,w}(v)$ denotes the total number of shortest path between u and w that pass through vertex v and $\sigma_{u\,w}$ denotes the total number of shortest paths between u and w.

In Figure 2.1, top three vertices with the highest betweenness centrality values are vertex 8 = 28, vertex 3 = 16.67, and vertex 6 = 16.67.

- **Clustering Coefficient**

Clustering coefficient is a measure of degree to which nodes in a graph tend to cluster together. It is calculated as the ratio of the edges between the neighbors of a vertex to the total possible connection between them. The higher the clustering coefficient it is more likely that a vertex is part of a dense module with closely interconnected dependencies. Formally, the clustering coefficient of a vertex v is as,

$$C_i = \frac{2n_i}{k_i(k_i - 1)}$$

Where n_i denotes the number of links connecting the k_i neighbors of vertex i to each other.

In Figure 2.1, top three vertices with highest clustering coefficient values are Vertex 2 = 1.0, Vertex 7 = 1.0, and Vertex 1 = 0.67.

2.1.2 Network Based Properties

- **Vertices**

Total number of vertices in a graph. There are total 10 vertices in the graph from Figure 2.1.

- **Edges**

Total number of edges in a graph. There are total 36 edges in the graph from Figure 2.1.

- **Degree Distribution**

The degree distribution is the probability distribution of degrees of the vertices over the network. Most scale free system like social and biological networks observe a power law based distribution [4] that is there are many vertices with low degree and the number of vertices exponentially go down as the degree increases. In Figure 2.1, degree of vertices are, deg (V1) = 4, deg (V2) = 3, deg (V3) = 5, deg (V4) = 6, deg (V5) = 4, deg (V6) = 5, deg (V7) = 3, deg (V8) = 3, deg (V9) = 2, deg (V10) = 1. Degree distribution is (d_1, d_2, …, d_{n-1}), where d_k is the number of vertices with degree k. Degree Distribution for graph in Figure 2.1 is (1, 1, 3, 2, 2, 1).

- **Shortest Path and Diameter**

Shortest path is a path between two vertices in a graph such that sum of weights of participating edges is minimized. The diameter of a graph is the largest value of all the shortest paths. In Figure 2.1, shortest path between vertex 1 and vertex 9 is 3 and diameter of a graph is 4, because that is the maximum value of all the shortest paths.

- **Articulation Point**

A vertex in a connected undirected graph is an articulation point if removal of that vertex and all edges incident to it result in a disconnected graph. Articulation points in a graph are critical to communication; all paths between certain vertices have to pass through articulation point. In Figure 2.1, vertex 8 and vertex 9 are the articulation points.

- **Modularity**

Modularity is a property of a network and a specific proposed division of that network into communities. Modularity in a network is computed as, $\sum_i (C_{ii} - a_i^2)$, where C_{ii} is the percentage of the number of edges per community C_i and a_i is the percentage of the edges connected to Community C_i.

Modularity of the graph in Figure 2.1 is 0.057 with the communities, Community 1: Vertex 1, 2, and 3, Community 2: Vertex 4, 5, 6, and 7, Community 3: Vertex 8, 9, and 10.

2.2 Brief Outline of Our Applications

2.2.1 Software Engineering

We can represent different versions of software systems as networks. The usage dependencies in each version can be modeled as a directed network, where vertices represent different modules in the software system and each edge (u, v) represents a dependency from module u to module v. We compute several graph properties for each network such as, in-degree and out-degree: which gives number of dependencies of a module in the software system, diameter of a network gives critical path in the system,

high betweenness centrality represents more calls to the module representing vertex in the system, articulation points represent important module in systems, etc. Our goal is to investigate several ways of measuring the amount of disruption by examining changes in combinatorial properties across the different software version.

Several researchers have also applied graph theory measures to study software systems. Myers [5] analyzed 6 software projects and found them to be scale-free, small-world networks. Chatzigeorgiou et al. [6] applied graph theory to detect design patterns, and improve coupling and cohesion. They performed a case study on three software systems and observed that software networks are scale-free. Wang et al. [7] conducted an analysis of 223 versions of the Linux kernel, and also observed these networks to be scale-free and satisfy small-world properties. Savic et al. [8] arrived at the similar conclusion in an analysis of 5 open source projects.

2.2.2 Bioinformatics

Microarray data analysis emerges in the decade as a key method for obtaining correlation among genotype and phenotype information. DNA microarray technology measures the gene expression level of thousand of genes under multiple experiment conditions [9]. This technology has been widely used in many areas of biology. It helps in the identification of new genes, and to understand their functioning and expression levels under different conditions. Microarray technology also helps researches to learn more about different diseases especially the study of cancer. It can also be used in the study of correlation between therapeutic responses to drugs and the genetic profiles of the patients, and impact of toxins on the cells and their passing on to the progeny. Large amount of data is produced in the microarray technology and it's very difficult to

understand such a large data. Proper analysis of the data is important to extract biologically relevant information. Microarray data can be represented as matrix where rows correspond to different genes and columns to experimental conditions. One important analysis of microarray data is the discovery of biclusters, which are groups of genes that show similar behavior across specific group of experimental conditions.

The term biclustering was first used by Cheng and Church [10] in gene expression data analysis. It is also referred as “direct clustering” [11], “box clustering” [12], “subspace clustering” [13], and “co-clustering” [14]. Biclustering problem has been shown to be NP-hard [11] [15], and almost all the approaches presented to date are heuristics. Many approaches for biclustering in expression data have been proposed. Several surveys about biclustering techniques have been published [16-18]. Some of the prominent biclustering methods are Cheng and Church [10], xMotifs [19], SAMBA [20], ISA [21], OPSMs [22], CPB [23], BiMax [24].

2.2.3 Community Detection

Community structure is a network characteristic describing the propensity of groups of vertices to form dense connection within the group than across the groups. This characteristic is used in the analysis of networks for many applications including hierarchies of organization [25], collaboration networks [26], protein interactions [27], and stability of electrical grids [28]. The problem of community detection involves finding such connected groups in a given network has become popular algorithm in recent years.

Newman and Girvan [29] proposed a greedy agglomerative approach based on maximization of modularity for hierarchical community detection. Clauset, Newman and Moore [30] proposed fast implementation of a previous technique proposed by Newman et al. [29]. Guimera and Amaral [31] proposed community detection algorithm based on exhaustive modularity optimization via simulated annealing. However, Modularity maximization fails to identify communities smaller than a certain scale, therefore bring a resolution limit on the communities detected by a pure modularity optimization approach. Blondel et al. [32] proposed new technique based on a local optimization of Newman and Girvan modularity in the neighborhood of each vertex. This algorithm solves resolution method problem due to the intrinsic multi level nature of the algorithm.

Tantipathananandh et al. [33] proposed an offline clustering framework based on finding optimal graph colorings. They presented heuristic algorithm which find near optimal solutions. Ning et al. [34] proposed an incremental algorithm which is initialized by a standard spectral clustering algorithm, followed by the updates of the spectral as the dataset evolves. Leung et al. [35] discussed the potential of the label propagation algorithm for dynamic network data. Mucha et al. [36] generalized the Laplacian dynamics approach to obtain a version of the modularity measure for multi slice (i.e. dynamic) networks.

2.3 Relating Graph Properties to Application Domains

The Table 2.1 presents the relation of different graph properties with the two application areas, software engineering, and bioinformatics. This provides an example of

how we can translate application characteristics into graph properties and use these properties to analyze the underlying systems.

Graph Property	**Software Systems**	**Biological Relevance**
Vertices	Modules in the software systems	Genes in gene expression matrix
Edges	Dependencies between modules in the software systems	Similarity between genes under an experimental condition
In-degree	Number of dependencies of a module in the software systems	Number of genes with the similar behavior under an experimental condition
Out-degree		
Diameter	Critical paths of the software systems	Critical path of the biological networks
Betweenness Centrality	High: the more calls to the module representing the vertex	In protein networks, it represents key connector proteins, i.e. bottlenecks, with particular functional properties
Clustering Coefficient	High: set of interdependent modules	High: set of interdependent genes
Articulation Point	Important module in the software systems	Important gene / protein in the biological network
Modularity	A high modularity indicates that the two groups of modules have high probability of belonging to same community	High value of modularity indicate the two groups of genes have high probability of belonging to same community

Table 2.1: *Relation of graph properties and application domains*

Chapter 3

Analysis of Software Networks

3.1 Introduction

Software maintenance consists of four parts, Corrective Maintenance, Adaptive Maintenance, Perfective Maintenance, and Preventive Maintenance [37]. Corrective maintenance is performed after a fault or problem emerges in a system with the goal of restoring the functionality of the system. Adaptive maintenance required to adapt the software to new environment. Perfective maintenance is the process of receiving requests for enhancement or modifications and implementing them. Finally, Preventive maintenance deals with updating documentation to make the software more maintainable. Corrective Maintenance is considered as 'traditional maintenance', while others are part of 'software evolution'.

Understanding the evolution of networks is an important analysis task. However, due to large number of components in real world systems, it is difficult to get a quick summary of network changes. In this section, we explore different combinatorial metrics to quantify the difference between networks. We are interested in measuring the amount of disruptions by examining changes in combinatorial properties across networks. We demonstrate the use of combinatorial properties in understanding the evolution of software system networks. It is important to understand the evolution of software systems for assessing their long term maintainability. Inter-class relationships play important role in object oriented systems. We are interested in quantifying the extent to which such

relationships are disrupted or preserved in the midst of software evolution [38]. We explore combinatorial metrics to quantify and evaluate the difference between networks representing several versions of JHotDraw software. Our results show that these statistics provide important insights in understanding how the JHotDraw code evolved over time.

3.2 Methodology

We used six versions of JHotDraw 5 [39] from March 2001 to January 2004. These are referred as Version 1 to Version 6 in this document. The specific versions are listed in Table 3.1. We extracted use relationships such as inheritance and implementation, method calls and class member access, object declaration and instantiation from each version using SPARS-J [40-41]. Next, we represented each version as a directed graph, where vertices represent classes from software code and each edge (u, v) is a dependency from class u to class v. Our objective is to find the evolutionary characteristics such as: points of significant change in the software and how these changes affect crucial classes in the network using combinatorial or graph based metrics.

We compute the values of the graph properties discussed in chapter 2 and their change in rankings to analyze these networks. We use the Matlab BGL library [42] to compute most of the properties. The communities are computed using a Matlab code based on the modularity maximization algorithm described in [30].

Version	Date	Files	Commit Messages
Version 1	3/9/2001	304	Merge to JHotDraw 5.2 (using JFC/Swing GUI components)
Version 2	10/24/2001	720	Before merge for version 5.3 (dnd, undo, …), merge dnd (before 5.3)
Version 3	8/4/2002	392	After various merges.. (before 5.4 release)
Version 4	11/8/2002	2	Refactor to use Standard Storage Format as a superclass
Version 5	5/8/2003	44	Refatoring of Cursor. – java.awt.Cursor (class) has been systematically replaced
Version 6	1/9/2004	484	After renaming the CH.ifa.draw to org. jhotdraw

***Table 3.1**: Commits with perfective changes in JHotDraw*

We measure the overall change in values and rankings of the vertices across different version by developing the following formulas,

$$Rank\ Disruption = \frac{\sum_{v \in All\ Vertices} | Rank_{i+1}^{v} - Rank_{i}^{v} |}{Total\ Vertex\ Number}$$

$$Value\ Disruption = \frac{\sum_{v \in All\ Vertices} | Value_{i+1}^{v} - Value_{i}^{v} |}{Max\ (Value_{i+1})}$$

Where, $Rank_i$, and $Value_i$ represent the rank and value of the corresponding property in version *i*.

3.3 Results and Analysis

(Text in this section is mostly paraphrased from our publication [43])

In this section, we present results of the combinatorial properties discussed in the section 3.2 and discuss how they provide us knowledge about the evolution of JHotDraw.

3.3.1 Network and Vertex Properties

The number of vertices in a network represents the number of classes in the network. As the versions evolve, some vertices are deleted and new ones are added. A comparison between the number of added, deleted and retained vertices in the network provides a rouge estimate of the difference between the versions. The number of edges in the network represents the dependencies in the software. Similar to vertices, as the versions evolve, some edges are deleted and new ones are added. A comparison between the number of added, deleted and retained edges across different versions gives an estimate of the scale of the evolution.

Table 3.2 presents the values of network based properties for six version of JHotDraw software. The highest and second highest changes in additions and deletion of vertices, edges and articulation points are shown in bold and italic respectively. A value of vertices and edges increase across the versions this indicates that network grows over the time. We see that major changes happen in Version 2 to Version 3 and Version 4 to Version 5, because all the bold and italic values are under Version 3 and Version 5 in Table 3.2. Diameter and average path length do not grow that much this indicate that the new classes are added together as interdependent modules to the periphery rather than individually scattered across the systems. Articulation point's increases version by version and this tells that in later versions there are more regions of potential disconnect.

The number of communities also increases version by version and it indicates that there are larger numbers of modules present in later versions. We also note that most of the vertices are concentrated amongst the top two communities, and most of the elements in consecutive communities are retained. The increase in communities is therefore due to the newly added vertices.

Property	**V1**	**V2**	**V3**	**V4**	**V5**	**V6**
Vertices	159	177	302	339	528	544
Add (Delete)	0 (0)	18 (0)	*125 (0)*	38 (1)	**190 (1)**	16 (0)
Edges	775	832	1454	1684	2136	2167
Add (Delete)	0 (0)	74 (17)	**655 (33)**	256 (26)	*466 (14)*	64 (33)
Articulation Points	7	8	26	33	104	105
Add (Delete)	0 (0)	1 (0)	*18 (0)*	7 (0)	**71 (0)**	1 (0)
Diameter	6	6	7	9	9	9
Average Path Length	2.27	2.29	2.54	2.7	3.4	3.3
Communities	6	5	9	10	20	19
Top Two Communities	112	139	211	233	335	304
Common Elements	0	.80	**.62**	.84	*.61*	.88

Table 3.2: *Network-Based properties of different versions of JHotDraw. The Add (Delete) rows correspond to the properties in the previous row. The highest change in rows 3, 5 and 7 is marked by bold and the second highest by italics.*

Figure 3.1 and 3.2 show the degree distribution of the in-degrees and out-degrees of the six versions. Both the distributions observe the power law based degree distribution, where the numbers of vertices per degree exponentially decrease the value of the degree. The in-degree distribution shows this property more prominently than the out-degree distribution. As per our previous findings, there is big change in Version 2 to Version 3 and Version 4 to Version 5 and out-degree distribution graph support that finding, as we clearly see similarity and difference between versions.

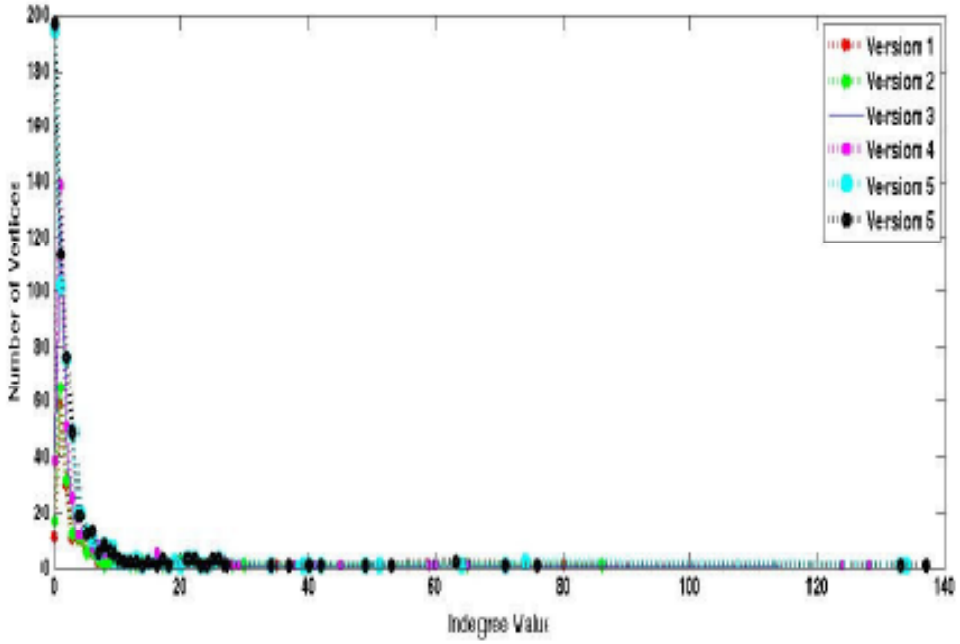


Figure 3.1: *In-degree Distribution across the six versions of JHotDraw*

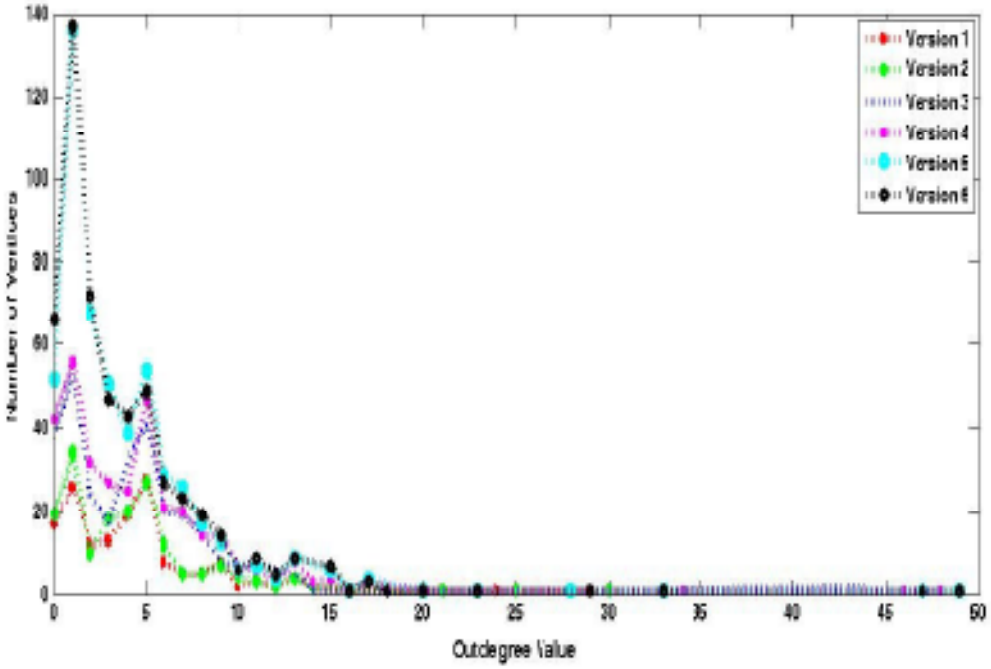


Figure 3.2: *Out-degree Distribution across the six versions of JHotDraw*

Table 3.3 shows the values of vertex-based properties of the network. It shows change in the value disruption and rank disruption values across six versions calculated using formulas mentioned in section 3.2. The highest and second highest changes are marked as bold and italic fonts respectively. Here also, we see that there is significant change in the evolution of Version 2 to Version 3 and Version 4 to Version 5. We also compare the top 25 highest ranked vertices for each property. Retained Vertices present the vertices that are common in the set of top 25 vertices for consecutive versions. Vertices in V_i only means vertices that are present in the set of top 25 in Version V_i but not in V_{i+1}. Similarly Vertices in V_{i+1} means vertices that are present in the top 25 in Version V_{i+1} but not in V_i. Newly added vertices refer to the vertices which are newly

added in V_{i+1} and present in top 25 highest ranked vertices. There is least number of retained vertices across the versions for clustering coefficient, which indicates once again that the changes involves adding a set of interdependent modules rather than adding modules separately to different parts of the software. There is no significant change for in-degree, out-degree and betweenness centrality in the highest ranked vertices. This shows that the critical paths of software are probably left unchanged.

Property	**V1 – V2**	**V2 – V3**	**V3 – V4**	**V4 – V5**	**V5 – V6**
In Degree					
Value Disruption	.0022	**.0138**	.0025	*.0083*	.0007
Rank Disruption	.014	**.252**	.06	*.112*	.016
Change in Set of Top 25 Vertices					
Retained Vertices	24	20	20	21	23
Vertices in V_i only	1	5	5	4	2
Vertices in V_{i+1} only	1	1	2	3	2
Newly Added Vertices	0	4	3	1	0
Out Degree					
Value Disruption	.0025	**.0213**	*.009*	.002	.002
Rank Disruption	0.45	**.292**	.069	*.209*	.009
Change in Set of Top 25 Vertices					
Retained Vertices	24	17	20	24	24
Vertices in V_i only	1	8	5	1	1
Vertices in V_{i+1} only	1	4	4	1	1
Newly Added Vertices	0	4	1	0	0

Betweenness Centrality					
Value Disruption	.0004	*.0027*	.0017	**.0107**	.0016
Rank Disruption	.051	**.286**	.074	*.212*	.012
Change in Set of Top 25 Vertices					
Retained Vertices	24	17	20	17	22
Vertices in V_i only	1	8	5	8	3
Vertices in V_{i+1} only	1	5	3	7	3
Newly Added Vertices	0	3	2	1	0
Clustering Coefficient					
Value Disruption	0	**.0088**	0	*.0056*	0
Rank Disruption	.078	**.370**	.074	*.157*	.021
Change in Set of Top 25 Vertices					
Retained Vertices	16	13	21	14	19
Vertices in V_i only	8	12	3	11	3
Vertices in V_{i+1} only	1	2	3	2	3
Newly Added Vertices	8	10	0	9	3

Table 3.3: *Change in vertex-based properties across different versions of JHotDraw. The table shows the disruption in values and rank. It also compares the set of the top (highest ranked) 25 vertices. The highest and second highest change in disruption is marked by bold and italic.*

Figure 3.3 shows the correlation between in-out degree and betweenness centrality. There is positive correlation between degree and betweenness centrality. Classes with high importance (high in-out degree) have high dependencies (high betweenness centrality). Figure 3.4 shows the correlation between clustering coefficient

and betweenness centrality. Unlike the correlation between degree and betweenness centrality, there is negative correlation between clustering coefficient and betweenness centrality. We see that betweenness centrality value increases due to increase in edges and vertices. However, clustering coefficient values do not increase. Once again this observation indicates that the newly added vertices are clusters of interdependent modules added at the end of the paths.

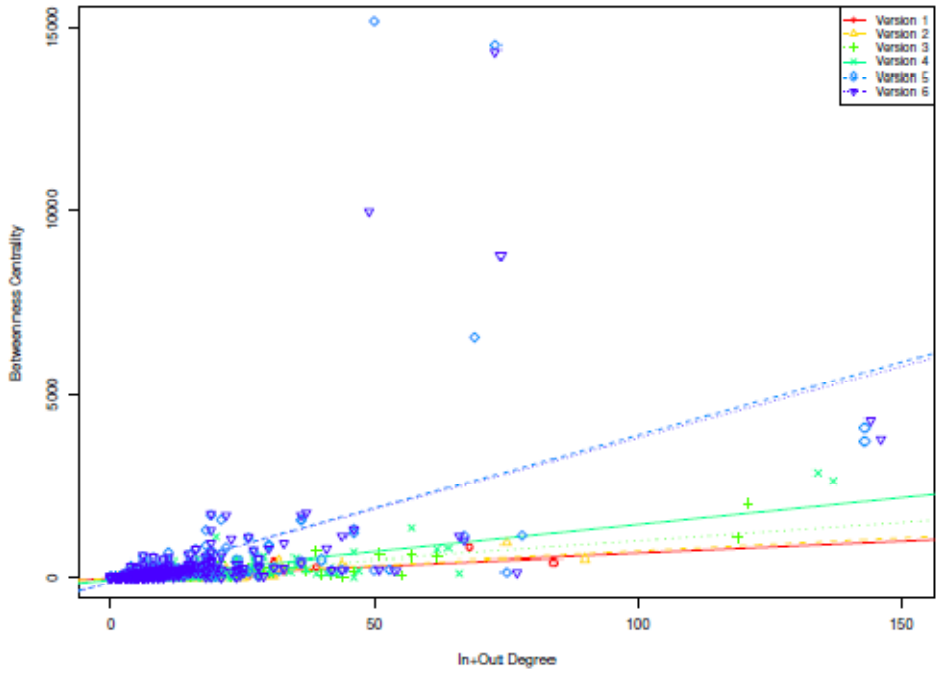


Figure 3.3: *Positive correlation between in-out degrees and betweenness centrality*

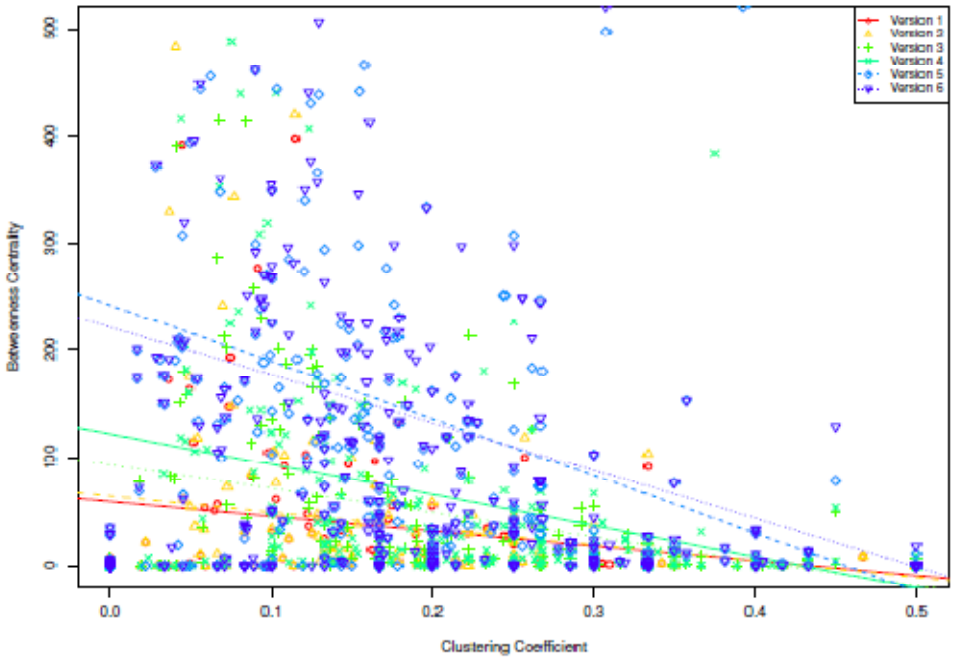
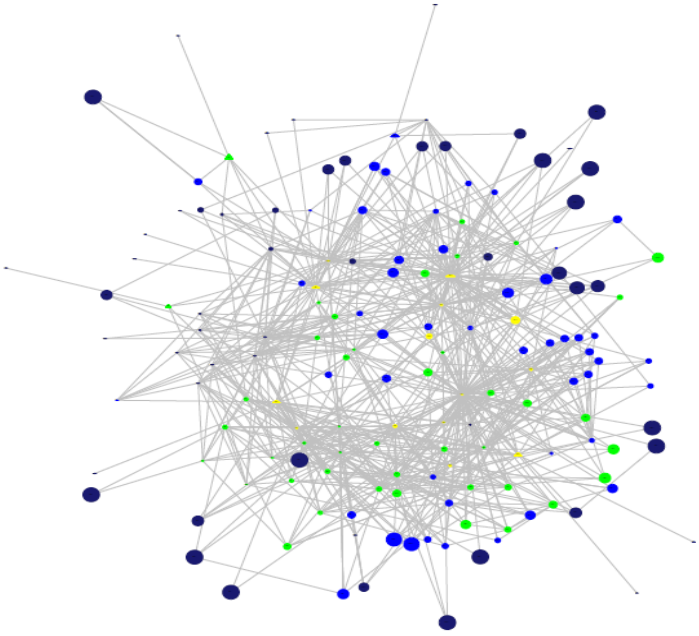
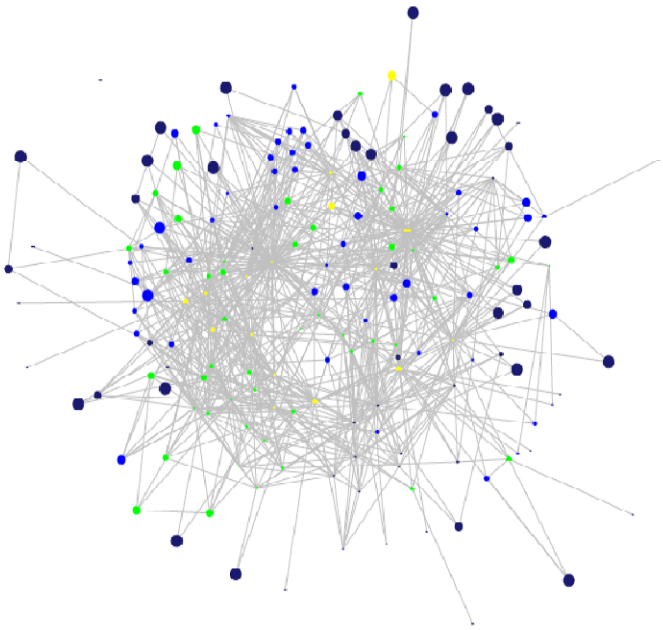


Figure 3.4: *Negative correlation between clustering coefficient and betweenness centrality. Note that this plot was clipped at y = 500 to highlight the correlation.*

Figure 3.5 shows the spring layout graphs of networks, Version 1 and Version 2 using GraphViz [44]. The vertex color and size represents the value of betweenness centrality and clustering coefficient respectively. The lighter color vertex indicates vertex with the high betweenness centrality value and the large size vertex represent the vertex with the high clustering coefficient value. We can see that there is negative correlation between clustering coefficient and betweenness centrality because the vertices at the peripheries are dark and larger in size. This also confirms our hypothesis that the newly added vertices are clusters of interdependent modules added at the end of the paths.



Version 1



Version 2

Figure 3.5*: Networks representing Version 1 and Version 2. Lighter vertices indicate high betweenness centrality. Larger vertices indicate high clustering coefficient*

3.3.2 Identifying Crucial Vertices

We divide the vertices into four groups; High, Extra High, Low and Extra Low. A vertex is classify as ‘High’, if it is in top 25 rank for at least one of the following categories; high in-degree, high out-degree, high betweenness centrality and high clustering coefficient. A vertex is marked as ‘Extra High’, if it is in top 25 rank for at least two categories listed above. On the other hand, a vertex is consider as ‘Low’, if it has zero value for any one of the categories and it is not listed as a ‘High’ vertex. A vertex is marked as an ‘Extra Low’, if it has zero value for betweenness centrality as well as clustering coefficient. All remaining vertices go into category ‘Other’. “Extra High’ and ‘High’ vertices represent important classes in the software on the other hand ‘Low’ and ‘Extra Low’ vertices represent classes which are not important. They are peripheral classes and do not have any significant impact on the software as a whole.

Figure 3.6 shows the percentage breakdown of all vertices in each category for all versions. We see that Version 1 - Version 2 show similar breakdown of vertices as does Version 3 – Version 4 and Version 5 – Version 6. This matches our previous observation that the major changes occurred between Version 2 to Version 3 and Version 4 to Version 5. Also, Version 1 and Version 2 have the largest number of ‘High’ and ‘Extra High’ vertices i.e. all important classes in the software are added in earlier versions of software. On the other hand, Version 5 and Version 6 have the largest number of ‘Low’ and ‘Extra Low’ vertices, which shows that as the software matures more peripheral functionalities are added.

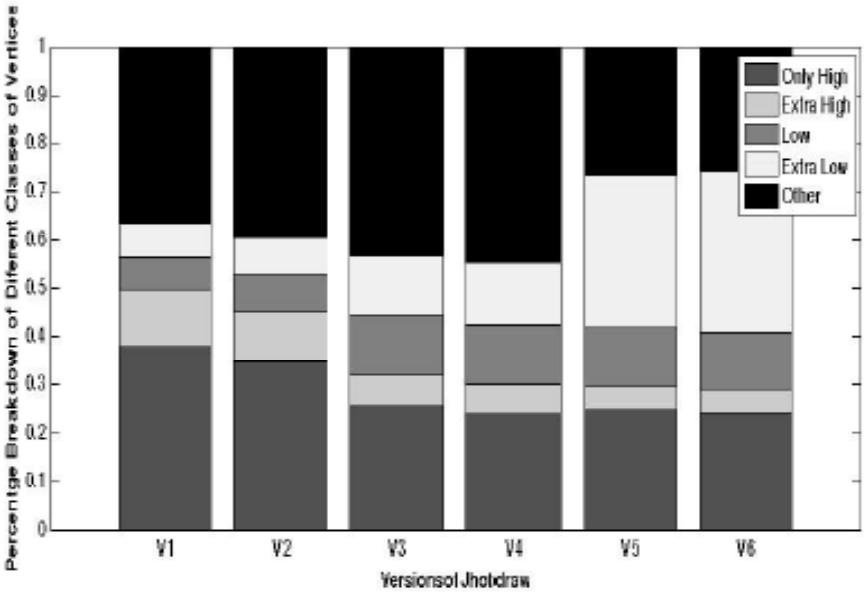


Figure 3.6: *Percentage breakdown of all vertices in each version*

3.3.3 Analysis of Newly Added Vertices

Figure 3.7 shows the classification of newly added vertices for each transition. In Version 1 to Version 2 transition, maximum percentage of newly added vertices are high clustering coefficients i.e. well connected modules have been added into Version 2. In transition from Version 4 to Version 5 and Version 5 to Version 6 most of the newly added vertices are zero betweenness centrality and zero clustering coefficient. Again, it confirms our previous finding that in later versions of software newly added vertices represent peripheral classes.

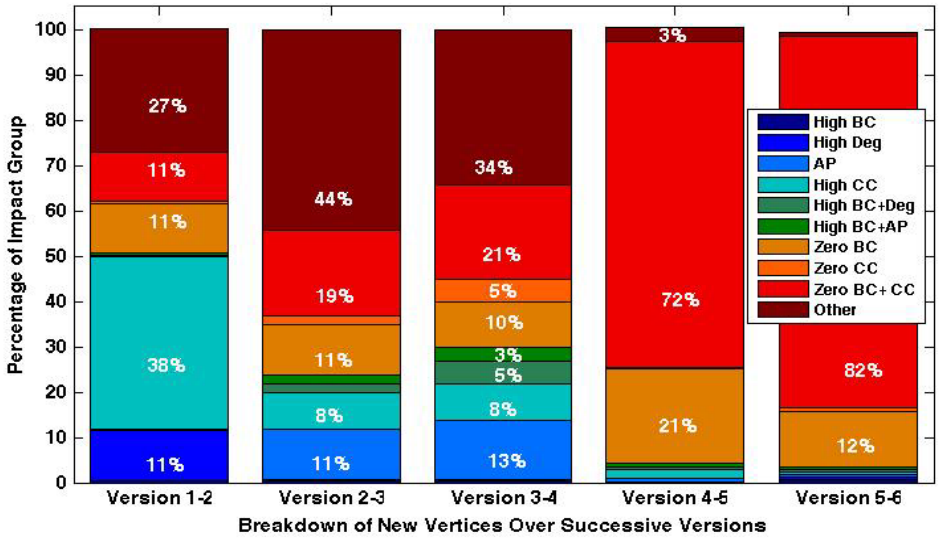


Figure 3.7: *Percentage of new vertices per impact group with respect to the total number of vertices added*

3.3.4 Analysis of Community Properties

In large networks, communities represent subset of the network with highly connected vertices. For software networks, identifying communities help in discovering the working architecture of the software system where the communities are aggregate components consisting of classes that interact highly with each other. We applied a community detection algorithm [30] to discover such aggregate components and to track the stability of these components over time. In Table 3.2, we see that the number of communities increases. We also note that, most of the vertices are concentrated amongst the top two communities and most of the elements in consecutive communities are retained. The community detection method, though extensively used is still heuristics and has some drawbacks such a resolution limit, i.e. can't find communities smaller than a certain size and sensitivity to tie-breakers, i.e. result can be significantly altered due to choices in tie-breaking [45]. In particular, later versions of the software have more

communities; most of the new communities have very few vertices (about two to three elements). Due to sensitivity of the algorithm these small communities are not meaningful and we therefore focus on the communities with larger membership (at least 8 members).

We note that each version has two large communities (over 50% of all of vertices) Table 3.4 compares these top two communities across all versions. We see that there is a large intersection between corresponding communities in consecutive versions, as indicated by the row 'Common Elements'. We find that vertices in these large communities tend to be retained from one version to the next. The fact that these tend to be stable across versions gives us confidence of the validity of the community detection algorithm. In particular, across all versions, two large communities seem to be centered on two key interfaces, 'draw.framework.Figure', the main interface for all figures, and 'draw.framework.DrawingView', the main interface for rendering drawings. A closer inspection across all versions indicates that one community has mostly figure and handler-related classes while the other has mostly drawing and toolbar-related classes. We observe that 'Figure' and 'DrawingView' are in the same package but ended up in different communities. Likewise, many detected communities cut across the hierarchical package structure, which seems to indicate that the working subset of classes are not confined to packages, but to some different aggregate. This hints at a potential division of classes for restructuring.

Property	V1-V2	V2-V3	V3-V4	V4-V5	V5-V6
Elements in V_i	112	139	211	233	335
Elements in V_{i+1}	139	211	233	335	304
Common Elements	112	132	197	204	269
Percentage w.r.t V_i	1	.94	.93	.87	.80
Percentage w.r.t V_{i+1}	.80	.62	.84	.61	.88

Table 3.4: *Analysis of similarities between large communities*

3.3.5 Impact on Quality

After each version, we looked at all changed files during the transition of that version. The number of file involved in each revision is counted and we looked for the keyword "bug fix" in each file. Table 3.5 shows bug frequency after each version. We can see that after Version 3 it has the highest number of bug fixes and second highest after Version 5. These intervals with the high percentage of bug fixes follow the periods with the highest measures of disruption (Version 2 to Version 3 and Version 4 to Version 5).

Interval	Total Files Changed	Bug Fixes	Percentage
Post Version 1	94	0	0.00%
Post Version 2	176	0	0.00%
Post Version 3	172	38	22.09%
Post Version 4	1720	120	6.98%
Post Version 5	50	6	12.00%
Post Version 6	89	1	1.12%

Table 3.5: *Bug Frequencies after Each Version*

3.4 Discussion

We have applied different combinatorial or graph-theory based metrics to study the evolution of networks representing JHotDraw 5 software. These metrics provide insight to understand disruption between versions. Our observations can be summarized as follows,

a. The significant evolutionary changes occur between Version 2 to Version 3 and Version 4 to Version 5.
b. Degree Distribution for all versions follows the power law an indication that these are scale free networks.
c. The network has grown cumulatively. Newer vertices tend to get added in the peripheries.
d. There is positive correlation between betweenness centrality and in-out degree. On the other hand there is negative correlation between betweenness centrality and clustering coefficient.
e. The top 25 rankings of vertices were generally stable across versions. This indicates stability in the design.
f. The bug frequency is higher after Version 3 and Version 5. The degree of disruption can help explain why bug incidence increases.
g. The top two communities contained the bulk of the vertices in each version. There was significant overlap between corresponding communities across consecutive versions.

From these observations, it appears the original design was maintained throughout the different versions. One of the important finding is the quantification of the amount of disruption caused by different versions of code. We also note that the bug incidence is higher after version 3 and 5. The degree of disruption can contribute to explaining why the bug increases.

Chapter 4

Analysis of Gene Expression Data

4.1 Introduction

Gene expression datasets are constructed in matrices, where each gene in a matrix corresponds to one row and each condition corresponds to one column. Each element in the matrix represents the expression level of a gene under a specific condition. There are number of methods for analyzing gene expression matrices, one of the most used methods is clustering such as hierarchical clustering [46], k-means clustering [47], etc. Clustering techniques use to group either genes (or conditions), such that genes (or conditions) of one group are similar to each other and different from other groups. Most of the clustering algorithms consider all the conditions to group genes and all the genes to group conditions. Traditional clustering algorithms have been successfully applied in many contexts. However, they suffer from some limitations in the analysis of large and heterogeneous collections of gene expression data. Standard clustering group genes (or conditions) based on global similarities in their expression profiles. However, due to large amount of diverse data, biologically related genes may not show similar behavior across all the conditions but in a subset of them. Also, traditional clustering generally set each gene in a single cluster, but many genes can be involved in different biological processes.

Biclustering techniques have been presented as an alternative approach to traditional clustering. It performs clustering on genes and conditions simultaneously in order to identify subsets of genes that display similar expression patterns across subset of

conditions and vice versa. In traditional clustering algorithms, cluster of genes is selected considering all the conditions and cluster of conditions is selected considering all the genes. However, in biclustering algorithms, cluster of genes is defined using subset of conditions. Similarly, cluster of conditions is defined using subset of genes. Figure 4.1 demonstrates the clustering and biclustering of a gene expression matrix. Clusters of genes (rows) (Figure 4.1 (a)) must contain all conditions (columns), and clusters of conditions (columns) (Figure 4.1 (b)) must contain all genes (rows). Biclusters (Figure 4.1 (c)) correspond to arbitrary subsets of genes (rows) and conditions (columns).

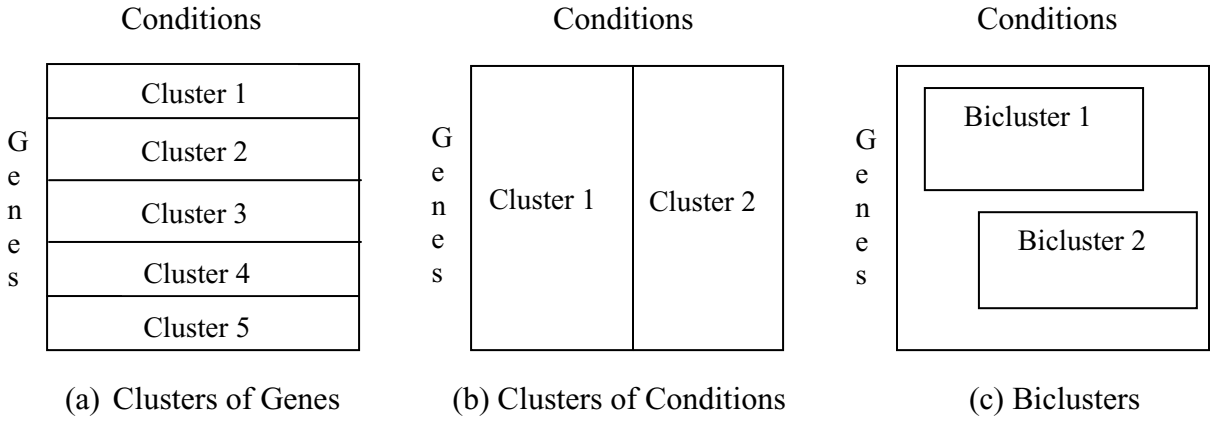


Figure 4.1: *Clustering and biclustering of a gene expression matrix*

4.2 Background

Consider gene expression data matrix, ‘A’ with set of rows ‘X’ and set of columns ‘Y’. Rows represent ‘n’ number of genes and columns represent ‘m’ number of conditions. Each cell of gene expression matrix represents expression level of gene under condition.

	Condition 1	Condition 2	…	…	Condition m
Gene 1	a_{11}	…	…	…	a_{1m}
Gene 2	…	…	…	…	…
…	…	...	…	…	…
…	…	…	…	…	…
Gene n	a_{n1}	…	…	…	a_{nm}

Table 4.1: *Gene Expression Data Matrix*

A cluster of rows (genes) is subset of rows (genes) that shows similar behavior across all the columns (conditions).

$$\textit{A cluster of rows (genes)} = (I, Y) \textit{ where, } I = \{i1, i2, \dots, ik\} \subseteq X \textit{ and } k <= n$$

Similarly, a cluster of columns (conditions) is subset of columns (conditions) that shows similar behavior across all the rows (genes).

$$\textit{A cluster of columns (conditions)} = (J, X) \textit{ where, } J = \{j1, j2, \dots, jk\} \subseteq Y \textit{ and } k <= m$$

On the other hand, a bicluster is a subset of rows (genes) that shows similar behavior across the subset of columns (conditions) and vice versa.

$$\textit{A bicluster} = (I, J) \textit{ where,}$$

$$I = \{i1, i2, \dots, ik\} \subseteq X, \textit{ and } k <= n,$$

$$J = \{j1, j2, \dots, jk\} \subseteq Y, \textit{ and } k <= m$$

So given a gene expression data matrix our goal is to identify different biclusters, such that each bicluster satisfies some specific characteristics of homogeneity.

Figure 4.2 illustrates types of biclusters proposed by Madeira et al. [16], They divided biclusters into four major classes, (i) Biclusters with constant values; where all the values are constant (Figure 4.2 (a)) (ii) Biclusters with constant values on rows or columns; where either rows or column values are constant (Figure 4.2 (b) (c)) (iii) Biclusters with coherent values; where each row and columns is obtained by addition or multiplication of the previous row and column by a constant value (Figure 4.2 (d) (e)) and (iv) Biclusters with coherent evolutions; where the direction of change of values is important rather than the coherence of the value. The first three categories are based on the actual numeric values of the data matrix and try to find subsets of rows and columns with similar behavior. The fourth category tries to find coherent behaviors regardless of exact numeric values in the data matrix. Each of these types of biclusters have different significant for discovering important knowledge from gene expression data.

Bozdag et al. [19] classifies biclustering patterns into two categories; (i) local pattern, and (ii) global pattern. A bicluster pattern is considers as local pattern, if it is defined on a single bicluster. All types of biclusters explained in the Figure 4.1 are come under local pattern, where no information is required about the elements outside the bicluster. On the other hand, in global pattern, the membership of a row (column) to a bicluster depends on the element of a row (column) external to the bicluster and/or on the membership of the row (column) to other biclusters.

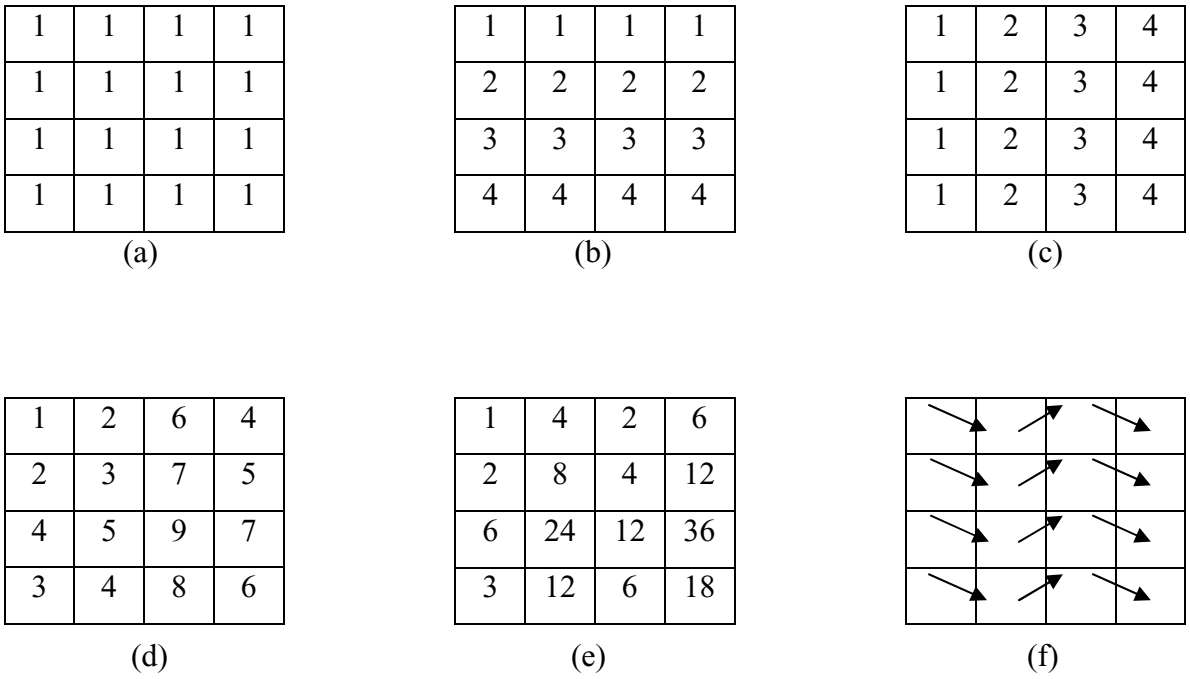


Figure 4.2: *Examples of different types of biclusters [16] (a) Constant bicluster, (b) Constant rows bicluster, (c) Constant columns bicluster, (d) Coherent values (Addictive model), (e) Coherent values (Multiplicative model), (f) Coherent evolutions bicluster*

In recent years, several algorithms have been proposed to find different types of biclusters. Some of the widely known algorithms include Cheng and Church [10], Iterative Search Algorithm [21], Correlated Pattern Biclusters [23], OPSM [22], xMotif [19], HARP [48], MSSRCC [49], SAMBA [20]. Most of the algorithms use greedy approach that start with either all rows or columns, and then iteratively eliminate them to optimize the objective function or they start with a random initial seed and use heuristics to converge to the final bicluster. Every biclustering algorithm focuses on few biclustering types shown in Figure 4.2. Cheng and Church algorithm finds constant values, constant rows and constant columns types of biclusters. HARP finds constant values and constant rows types of biclusters but not other types of biclusters. xMotif is

meant to find biclusters with constant columns. Some of the biclustering algorithms address the problem of finding coherent evolutions across the rows and/or columns of the data matrix regardless of their exact values. OPSM is designed to find coherent trends of up-down regulations in biclusters. Similarly, Correlated Pattern Biclusters algorithm focuses on biclusters with coherent evolutions. Cheng and Church, OPSM, HARP and Correlated Pattern Bicluster algorithms discover 'local patterns', while MSSRCC and SAMBA algorithms discover 'global pattern'. Some algorithms are designed to find overlapping biclusters, for e.g. Iterative Search Algorithm, SAMBA, and OPSM.

4.3 Our Contribution

We propose new biclustering algorithm which is based on the technique similar to graph alignment. Graph alignment is the problem of finding similarities between the structures of two or more graphs. Graph alignment is analogous to sequence alignments between genomes. Alignment in biological networks is very useful in bioinformatics research. Graph Aligner (GRAAL) [50] is one of the widely used algorithms for graph alignment. This algorithm is based on the network topology, which is the shape or structure of a network. GRAAL aligns pairs of vertices from different network based on their graphlet degree signature similarities [51], where a higher signature similarity between two vertices corresponds to higher topological similarity between their neighborhoods. GRAAL produces a global network alignment i.e. it aligns each vertex in smaller network to exactly one vertex in larger network. Thus, they do not allow gaps in alignments i.e. vertices without alignment in smaller network. Instead of finding alignment for all the vertices in smaller network with the larger network, we try to find

similar vertices between two networks using combinatorial properties, such as Clustering Coefficient, Betweenness Centrality, etc. (explained in Chapter 2). We divide our algorithm into three steps;

Step 1: Graph Representation of Gene Expression Matrix

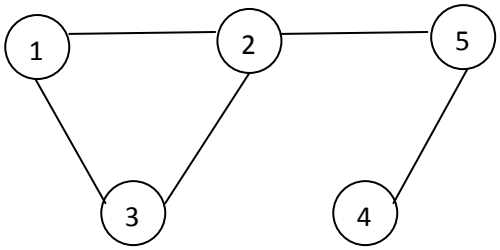
We represent gene expression data matrix in a graph format by creating a graph for each condition. So, we get N number of graphs where N is the number of conditions in a given input gene expression data matrix. In each undirected graph $G = (V, E)$, vertices (V) represent genes and edges (E) represent connections between genes according to the similarity criteria. There is an edge between two genes, if they show similar behavior under that condition. According to dataset, we set threshold value to decide the edge between genes. If the distance between expression values of two genes (e.g. G1 and G2) under that condition (e.g. C1) is less than the threshold value (Neighbor Threshold Value), then we add edge between G1 and G2 to the graph which is related to C1. There is an edge between two genes in a graph if,

$$Distance\ between\ two\ genes\ <=\ Neighbor\ Threshold\ Value$$

Figure 4.2 demonstrates how we represent gene expression matrix in a graph format. Figure 4.2 (a) is a sample gene expression matrix with 5 genes and N conditions. Figure 4.2 (b) is a graph representation of Condition 1 of gene expression matrix shown in figure 4.2 (a). In that graph there is an edge between vertex 1 (Gene 1) and vertex 2 (Gene 2) because the distance between Gene 1 and Gene 2 is less than 2 (assume 'Neighbor Threshold' value is 2). We apply similar procedure for all other conditions of gene expression matrix.

	C_1	C_2	…	…	C_n
G_1	10	…	…	…	a_{1n}
G_2	8	…	…	…	…
G_3	10	...	…	…	…
G_4	4	…	…	…	…
G_5	6	…	…	…	A_{mn}

(a) Gene Expression Matrix



(b) Graph representation for Condition 1 (Neighbor Threshold Value = 2)

Figure 4.3: *Example of graph representation of gene expression matrix*

Step 2: Similar Vertices (Genes) between Graphs

Next, we find the similar vertices between different graphs. First, we compute a difference matrix D of differences between vertices in two graphs. Rows of D correspond to vertices in Graph 1 and columns correspond to vertices in Graph 2. When computing the differences between a vertex u from Graph 1 with a vertex v in Graph 2, we consider clustering coefficient and betweenness centrality values (explained in the Chapter 2) of all the vertices. The higher the clustering coefficient it is more likely that a vertex is a part of dense module with closely interconnected components and betweenness centrality of a vertex represents how often it occurs in dependency path.

We calculate the clustering coefficient and betweenness centrality value for every vertex in Graph 1 and Graph 2.

$Clustering\ Coefficient\ (CC)\ Difference\ (i,j)$ is calculated as,

$$CC\ value\ of\ vetex\ i\ from\ Graph\ 1 - CC\ value\ of\ vetex\ j\ from\ Graph\ 2$$

$Betweenness\ Centrality\ (BC)\ Difference\ (i,j)$ is calculated as,

$$BC\ value\ of\ vetex\ i\ from\ Graph\ 1 - BC\ value\ of\ vetex\ j\ from\ Graph\ 2$$

Then, we compare all the vertices of Graph 1 and Graph 2 to find the difference between clustering coefficient and betweenness centrality values and calculate the matrix D.

$$Difference\ Matrix\ (i,j) = CC\ Difference\ (i,j) + BC\ Difference\ (i,j)$$

We consider '$vertex\ i$' of Graph 1 is similar to '$vertex\ j$' of Graph 2, if

$$Difference\ Matrix\ (i,j) <= Similarity\ Threshold\ Value$$

Using above procedure, we find the similar vertices between two graphs. First step gives us N graphs (N is the total number of conditions in gene expression matrix: G_1, G_2…, and G_n). We can compare all the graphs by two ways in order to find similar vertices between them, (i) compare G_1 and G_2, G_2 and G_3 and so on, and (ii) compare G_1 and G_2, G_1 and G_3, …, G_1 and G_n, G_2, and G_3, …, G_2 and G_n and so on. At the end of this step, we get similar vertices between different graphs i.e. pair of genes which shows similar behavior under conditions.

Step 3: Finding Biclusters

In the previous step, we get the similar vertices across different graphs; using that information in this step, we find different biclusters. Assume, we get the following similar vertices in different graph comparisons, In Comparison of Graph 1 and Graph 2; similar vertices are V1 – V2, V4 – V8, V5 – V7, etc.

Comparison 1 (Graph 1 and Graph 2): V1 – V2, **V4 – V8**, **V5 – V7**, …

Comparison 2 (Graph 4 and Graph 5): V7 – V9, **V4 – V8**, V1 – V10, **V5 – V7**, …

Comparison 3 (Graph 5 and Graph 6): **V4 – V8**, V15 – V16, **V5 – V7**, V11 – V22, …

Comparison 4 (Graph 8 and Graph 9): V1 – V3, **V4 – V8**, **V5 – V7**, …

In this step, we find the common pairs of similar vertices between different graph comparisons. In above example, we get following common vertices in different graphs,

Common Vertices (Genes): 4, 8, 5, And 7 in

Graphs (Conditions): 1, 2, 4, 5, 6, 8, And 9

In our graph representation of gene expression matrix, vertices represent genes and graphs represent the conditions. Therefore, we get the set of vertices (Genes), which are common across different graphs (set of conditions). In other words, we get the set of genes which shows similar behavior across set of conditions. Similarly, we find all common pairs of similar vertices between different graph comparisons. Then, we filter

the results by applying two conditions (i) minimum number of genes in a bicluster, and (ii) minimum number conditions in a bicluster.

4.4 Results and Assessment

Many Biclustering algorithms produce different results for same gene expression datasets. Moreover, same algorithm produces different results for different parameter settings. One of the important things in deciding the better algorithm is to check correctness of the results. All validations techniques of traditional clustering algorithms can be divided in to two types; internal validation measures and external validation measures [52]. Internal validation techniques are based on the data intrinsic to the data alone; they don't use additional knowledge in the form of true clusters. On the other hand, external validation measures evaluate clustering results based on the correct clusters. In cases where true clusters are not available internal validation measure is useful. In most of the biclustering papers, external validation measures have been used to evaluate the results. Most of them recommend external validation measures because it is not clear how to extend notions such as homogeneity and separation to the biclustering context [53] and there are some issues with internal validation measures [52-53]. We used two types of datasets to test our algorithm, (i) synthetic datasets, and (ii) real datasets. For synthetic datasets, we used external validation techniques, because we already knew the true results, and for real datasets, where we did not know the true biclusters, we used internal validation techniques to validate the results.

4.4.1 Synthetic Datasets

We generated synthetic datasets by implanting fixed size biclusters in matrices. All matrices are of size 50 rows (genes), and 50 columns (conditions). We implanted 10 non-overlapping biclusters (with four genes and four conditions each) in every matrix with no noise. We created three different matrices for three types, namely Constant Biclusters, Constant Rows Biclusters, and Constant Columns Biclusters. Matrix 1 has 10 implanted biclusters of type constant biclusters, matrix 2 has 10 implanted biclusters of type constant rows biclusters, and matrix 3 has 10 implanted biclusters of type constant columns bicluster.

In order to validate the bicluster results, Prelic et al. [54] have used following gene match score formulae, which reflects the average of the maximum match scores for all biclusters in M_1 with respect to the biclusters in M_2

$$S\,(M1, M2) = \frac{1}{|\,M1\,|} \sum_{(G1,C1)\in M1} max_{(G2,C2)\in M2}\ \frac{|\,G1\ \cap G2\,|}{|\,G1\ \cup G2\,|}$$

In biclustering, genes as well as conditions play an important role. We need to check genes as well as conditions to validate the biclustering results. So, we added Jaccard coefficient [56] score of conditions $\frac{|\,C1\ \cap C2\,|}{|\,C1\ \cup C2\,|}$ in above gene match score formula. Instead of only Jaccard coefficient score of genes, we took the average of Jaccard Coefficient score for genes and conditions, i.e. $avg\ \left(\frac{|\,G1\ \cap G2\,|}{|\,G1\ \cup G2\,|}\ and\ \frac{|\,C1\ \cap C2\,|}{|\,C1\ \cup C2\,|}\right)$.

$$S\,(M1, M2) = \frac{1}{|\,M1\,|} \sum_{(G1,C1)\in M1} max_{(G2,C2)\in M2}\ avg\ \left(\frac{|\,G1\ \cap G2\,|}{|\,G1\ \cup G2\,|}\ and\ \frac{|\,C1\ \cap C2\,|}{|\,C1\ \cup C2\,|}\right)$$

Consider M_1 = Result of an algorithm and M_2 = True Result, then we calculate two scores, both scores take the maximum value of 1, if both the results are similar ($M_1 = M_2$).

$$Score\ 1: S\ (M1, M2)$$

$$i.e. How\ much\ percentage\ of\ the\ algorithm's\ result\ is\ included\ in\ the\ true\ results$$

$$Score\ 2: S\ (M2, M1)$$

$$i.e. How\ much\ percentage\ of\ the\ true\ results\ is\ included\ in\ the\ algorithm's\ results$$

Table 4.2 shows results for synthetic datasets. All biclusters with constant values, constant rows, and constant columns found by our method, because both the scores for all three matrices are 1,

Dataset	Number of Biclusters Found	Score 1	Score 2
Matrix 1 (Constant)	10	1	1
Matrix 2 (Constant Rows)	10	1	1
Matrix 3 (Constant Columns)	10	1	1

Table 4.2: *Results on synthetic dataset*

4.4.2 Real Datasets

We used two real datasets (sample datasets from "Biclustering Analysis Toolbox V2.2" [55]) to test on our algorithm. Table 4.3 shows description of these two datasets. Dataset_1 has 34 genes, 153 conditions and Dataset_2 has 419 genes and 70 conditions.

Dataset	Number of Genes	Number of Samples
Dataset_1	34	153
Dataset_2	419	70

Table 4.3: *Real Dataset Description*

Here, we did not know the true biclusters, so we decided to use internal validation measures. As a first step, we calculated compactness of the bicluster. It measures how closely related the objects in a bicluster are. There are different measures estimate the cluster compactness based on the distance, such as maximum or average pair wise distance, and maximum or average center-based distance. We calculated the Euclidean distance [57] between each pair of conditions in a bicluster, and found the maximum distance between conditions in a bicluster.

$$Euclidean\ distance\ (C1, C2) = \sqrt{\sum_{i=1}^{m} (C2_i - C1_i)^2}$$

Then, we calculated average maximum distance between two conditions in a bicluster for all biclusters in a result. We compare our results with Cheng and Church, Iterative Search Algorithm, OPSM and BiMax biclustering algorithms (using BicAT Analysis Toolbox [55]). If we consider the constant biclusters, constant rows biclusters and constant columns biclusters, then the distance between any two samples should be zero. Table 4.4 and 4.5 summarize the results for real datasets. We see that, our algorithm gives better results for constant biclusters. Table 4.4 shows that the average maximum distance score for Dataset_1 is very low for our algorithm followed by Cheng and Church, Iterative Search Algorithm, BiMax and OPSM. Similarly, Table 4.5 shows that

the average maximum distance score for Dataset_2 is very low for our algorithm followed by BiMax, Iterative Search Algorithm, Cheng and Church, and OPSM. Average maximum distance score is very high for some of the algorithms such as OPSM, BiMax etc. because, they focus on biclusters with coherent evolutions and compactness is not the right validation measure for such type of biclusters. We may get very high maximum distance between samples for biclusters with coherent evolutions.

Algorithm Name	**Parameter Settings**	**Number of Biclusters**	**Avg max distance between two conditions in a Bicluster**
Our Algorithm	P1: 7, P2: 0; P3: 4, P4: 4	11	134.01
	P1: 3, P2: 0; P3: 2, P4: 2	16	67.74
Cheng and Church		10	109.54
Iterative Search Algorithm		3	7011.71
OPSM		10	9392.10
BiMax		26	8718.92

Table 4.4: *Biclustering results for Dataset_1. (P1: Neighbor Threshold, P2: Similarity Threshold, P3: Minimum Number of Genes, and P4: Minimum Number of Conditions)*

Algorithm Name	**Parameter Settings**	**Number of Biclusters**	**Avg max distance between two conditions in a Bicluster**
Our Algorithm	P1: 0.5, P2: 0; P3: 4, P4: 4	722	3.08
	P1: 0.4, P2: 0; P3: 2, P4: 2	1298	2.55
Cheng and Church		10	17.41
Iterative Search Algorithm		38	11.54
OPSM		12	22.83
BiMax		1938	4.59

Table 4.5: *Biclustering results for Dataset_2. (P1: Neighbor Threshold, P2: Similarity Threshold, P3: Minimum Number of Genes, and P4: Minimum Number of Conditions)*

4.5 Discussion

We have used different approach than other previous biclustering algorithms for finding biclusters in a gene expression data. As we discussed earlier, most of the algorithms either start with both all rows and columns or start with random initial seed and then use greedy approach to find the biclusters. Therefore, the method is unable to search the space of all possibilities exhaustively. The structure of our method makes it possible to search every possible biclusters. Our primary results show that, our method is very good to find constant biclusters. Also, this approach looks promising to find other types of biclusters.

Our method of finding biclusters in a gene expression matrix has a vast scope for improvements and advancements. The process of finding similar vertices in two networks can be improve using combination of several combinatorial properties instead of using only clustering coefficient, and betweenness centrality. This will help for finding biclusters of type coherent values, and coherent evaluation. Also, we can improve this method to find overlapping biclusters.

Chapter 5

Efficient Algorithm to Finding Communities in Dynamic Networks

5.1 Community Detection

Community detection in a networks concerns collecting similar objects under one group. Objects in the same community are similar to each other and different from objects in the other communities. Two common methods for community detection are, divisive and agglomerative. Divisive method is a "top down" approach where initially all objects are in one community and then they divided into communities according to a similarity measure. Agglomerative method is a "bottom up" approach where at first each object is in its own community and then pairs of communities are merged according to similarity measure. A dendrogram, a branching diagram, represents the hierarchy of connections in the agglomerative method. In the network, we group vertices according to the edge structure such that there are many edges within the group and very few between the groups. We get densely connected components of the graph by applying community detection algorithm on them.

A popular method for community detection is based on maximization of a metric known as modularity proposed by Newman and Girvan [29]. Clauset, Newman and Moore [30] proposed an algorithm (this algorithm will be referred as "CNM" in this document) to efficiently obtain high modularity in large networks. This algorithm uses

greedy approach, where each vertex is in its own community followed by repeatedly join two communities whose amalgamation produces the maximum increase in modularity.

5.2 CNM Algorithm / Static Community Detection Algorithm

CNM algorithm uses modularity property of a network to find the communities. Modularity is a property of a network and measures the difference between the edges present within a group of vertices to the edges expected from random connections between them. The difference is normalized over the total number of edges in the graph. A high modularity indicates that the two groups have high probability of belonging to same community. Initially each vertex is assigned to a single community. Two communities are merged if the operation maximizes the increase in the total modularity of a network. The CNM algorithm proceeds in iterative steps combining communities until the potential increase of modularity becomes negative. The final set of communities is then identified as the closely connected groups of vertices in the network.

Let, A_{vw} be an element of the adjacency matrix of the network,

$$A_{vw} = \begin{cases} 1 & If\ vertices\ v\ and\ w\ are\ connected \\ 0 & Otherwise \end{cases}$$

Suppose the vertices are divided into communities such that vertex v belongs to community c_v. Then the fraction of edges that fall within communities, i.e., that connect vertices that both lie in the same community, is

$$\frac{\sum_{vw} A_{vw}\delta(C_v, C_w)}{\sum_{vw} A_{vw}} = \frac{1}{2m}\sum_{vw} A_{vw}\delta(C_v, C_w)$$

Where the δ-function $\delta(i,j)$ is 1 if $i = j$ and 0 otherwise and $m = \frac{1}{2}\sum_{vw} A_{vw}$ is the number of edges in the graph.

This quantity will be large for good divisions of the network, in the sense of having many within community edges, but it is not, on its own, a good measure of community structure since it takes its largest value of 1 in the trivial case where all vertices belong to a single community. However, if we subtract from it the expected value of the same quantity in the case of a randomized network, we do get a useful measure. The degree k_v of a vertex v is defined to be the number of edges incident upon it

$$k_v = \sum_w A_{vw}$$

The probability of an edge existing between vertices v and w if connections are made at random but respecting vertex degrees is $\frac{k_v k_w}{2m}$. Modularity Q is

$$Q = \frac{1}{2m} \sum_{vw} \left[A_{vw} - \frac{k_v k_w}{2m} \right] \delta(C_v, C_w)$$

The algorithm uses a greedy optimization in which, starting with each vertex being the sole member of a community of one, we repeatedly join together the two communities whose amalgamation produces the largest increase in Q. We continue this process until we get negative maximum Q value.

The algorithm has running time $O(md \log n)$ for a network with 'n' vertices and 'm' edges and a depth, 'd' of the hierarchical community structure and is thus known to perform efficiently on vertices up to 500,000 vertices [58].

Most community detection algorithms focus on finding the communities in static i.e. non-evolving networks. However, most real world networks such as social networks etc. evolve with the time. Networks change at each time step and most of the community detection algorithm consider each step as a separate network. The information regarding communities from the previous network is not used and communities have to recompute as a whole. To run the community detection algorithm on the complete graph for even a small change would be computationally very expensive. The efficiency of these algorithms can be greatly improved if the re-computation is limited only to the portions of the network that are affected by change. We propose a fast community detection algorithm for real-time dynamic networks that take advantage of community information computed in previous time steps. Our algorithm increases efficiency of the detected community structure because of using community information from previous time step networks.

5.3 Our Contribution / Dynamic Community Detection Algorithm

(Text in this section is mostly paraphrased from our publication [59])

We propose a community detection algorithm for dynamic networks which changes over time. Changes in the network involve addition or deletion of edges in the network. Our algorithm is based on the greedy agglomerative technique of the CNM algorithm. If the total numbers of edges in the network are sufficiently large, then a small change in the number of edges would not affect the fraction of edges in the graph, i.e. the values of C_{ij}. Therefore, we first apply the CNM algorithm on the initial network configuration and record each combination step, i.e. two communities that have merged

and value of increase in modularity due to merge. In every change in the network i.e. addition or deletion of edges in the network, we define the two vertices associated with the modified edge as being perturbed. The combination steps are replicated without any recalculation if participating vertices are not perturbed. Once we get first perturbed vertex in the combination step we switch back to CNM algorithm and continue until all the communities have been identified. Given a modified edge (a, b); replicate the combination steps of the previous time steps until vertex 'a' or vertex 'b' is encountered. Then switch back to the original agglomerative algorithm continue as in the static case.

Pseudo code for Dynamic Community Detection Algorithm

Input: Network G_0 and list of modified edges over time steps where t = 1, …, T.

Output: Community structure at time steps t = 1, …, T.

Steps:

1. The community structure of the input network G0 is initialized using the original greedy agglomerative algorithm
2. Each combination step is stored as a triplet <i, j, dQ>, t = 0, where i and j are communities that have merged and dQ is the increase in modularity due to merge.
3. For iterations over time steps t = 1, …, T
    a. Obtain change in edges. Let a and b be the vertices involved in the edge change
    b. Update network G_{t-1} to G_1 to include the change

c. Replicate combination steps of G_{t-1} until vertex a or b is encountered

d. Revert to original agglomerative algorithm

e. Continue until increase of modularity, dQ is negative

f. Delete combination steps for G_{t-1}

g. Store all the combination steps G_t

4. End

Data Structure for Dynamic Networks (CSR format)

Graphs can be represented as an adjacency matrix where rows and columns are labeled by graph vertices and value of adjacency matrix (V_i, V_j) is 1 if there is an edge between vertex V_i and vertex V_j otherwise 0. Adjacency matrix of large graphs is usually sparse matrix where most of the matrix values are zeros. Data structures for dynamic networks include adjacency lists, such as those used in [60], which are easy to modify through addition and deletion of elements to the list. However, adjacency list can potentially occupy non-contiguous addresses; it is not efficient memory utilization.

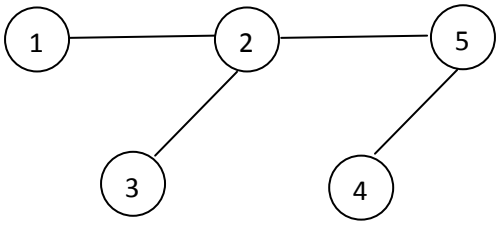
Compressed row storage method [61] is a popular format for representing sparse matrices. This method stores the non-zero elements of a sparse matrix into a linear array. In this method all the information about sparse matrix is stored into three vectors as described below,

a. **Values:** stores the non-zero values of a sparse matrix by walking down each column and writing a non-zero values
b. **Columns:** Value of Columns[i] is the number of the column of adjacency matrix that contains the Values[i] element.

c. **RowIndex:** Value of RowIndex[i] gives the index of the element of the Values array of the first non-zero element in a row 'i' of adjacency matrix.

The example of compressed row storage format for storing small graph is shown in Figure 5.1

In our implementation we used the compressed row storage method with few modifications. To identify the community structure, we added an extra vector C_ID to the original storage format. C_ID vector stores the community ID for each vertex. When an edge is deleted, the corresponding value is set to zero; when an edge is added, a new entry and value is added to the existing arrays. The advantages of this modified data structure are high cache utilization and easy to implement. However, due to addition of edges and deleted edges are represented by zeros the network tends to become larger as the number of modification increases.



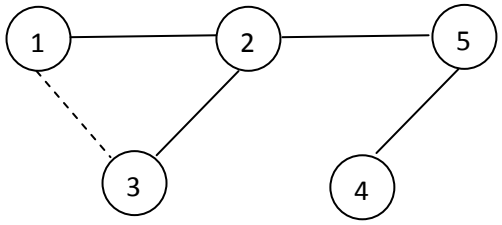
a. Network of 5 Vertices

	1	2	3	4	5
1	-	W1	-	-	-
2	W2	-	W3	-	W4
3	-	W5	-	-	-
4	-	-	-	-	W6
5	-	W7	-	W8	-

b. Adjacency matrix for the network

C_ID	1	2	3	4	5			
Index	1	2	5	6	7	9		
Columns	2	1	3	5	2	5	2	4
Values	W1	W2	W3	W4	W5	W6	W7	W8

c. CSR format for the original network



d. Original network with edge (1, 3) added

C_ID	1	2	3	4	5	**1**	**3**			
Index	1	2	5	6	7	9	**10**	**11**		
Columns	2	1	3	5	2	5	2	4	**3**	**1**
Values	W1	W2	W3	W4	W5	W6	W7	W8	**W9**	**W10**

e. CSR format for the modified network

Figure 5.1: *The CSR Format for a network. a) The original network. b) The sparse adjacency matrix corresponding to the network. The values represent the increase in modularity if the row and column are to be merged. c) The CSR format for the sparse matrix. d) The original network with new edge (1, 3) added. e) Modification to the original sparse matrix to add entries for edge (1, 3) and (3, 1).*

5.4 Results

In this section, we describe the results of our dynamic community detection algorithm method on a publicly available dataset on scientific collaboration. We use dynamic network data where all changes are available a priori and they are processed one change at a time.

DBLP database presents information on computer science publications listed in the DBLP Computer Science Bibliography [62]. The data in this dataset provides a snapshot of the bibliography as of April 12, 2006. The DBLP dataset maps each entry in the original DBLP data to one of six types of objects representing different types of publications. It includes links from publications to their authors and editors and from papers to the journal, proceedings, or book in which they appear, as well as citation links from one publication to another. From this data, we derive a dynamic co-authorship network for year 2000 and 2001 to test our algorithm. Both the networks (Year 2000 and 2001) have 3252 vertices. Network for year 2000 has 10997 edges and network for year 2001 has 11159 edges. Vertices in the network represent authors and edges represent co-authorship.

Network Name	Vertices	Edges
Year 2000	3252	10997
Year 2001	3252	11159

Table 5.1: *Network Information*

There are 2169 changes in the edges (1124 additions and 1044 deletions) from network of year 2000 to year 2001. We experiment with multiple changes at a time from 1 change at a time to 2, 4, 8, and 10 changes at a time. We compare the time required for

our dynamic algorithm and static algorithm at each step. We alternate the modifications between addition and deletion of edges to satisfy our assumption that small number in the number of edges would not affect the value of C_{ij}. However, in the DBLP dataset for year 2000 and 2001, there are 80 more additions than deletions, i.e. 2% change in the number of edges in the network. This change does slightly affect results during the final modification. The results and observations of our experiments are described below,

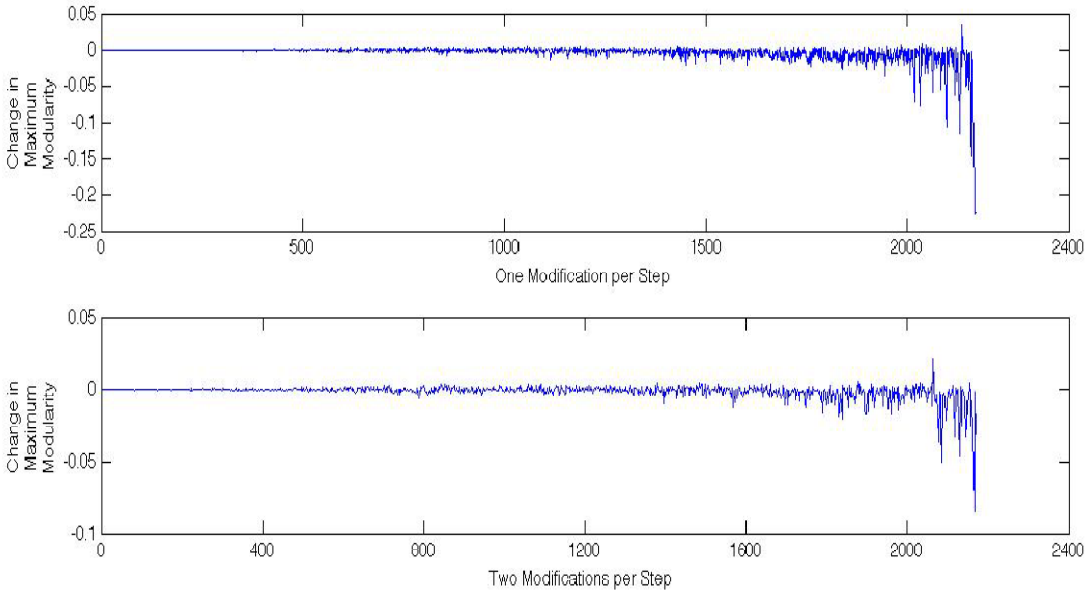


Figure 5.2: *Difference in maximum modularity of the static and dynamic method over each network. The X-axis plots the number of modifications and the Y-axis plots the difference in the modularity. Top: One change per time step and Bottom: Two changes per time step*

In order to find the correctness of our dynamic algorithm, we compare the maximum modularity obtained by our dynamic algorithm with the original static algorithm at each step. We can see in the Figure 5.2 and 5.3, the maximum modularity

obtained by two algorithms remain nearly same except the final few modification steps, where they diverge. The difference in the modularity between two methods during final steps is generally within 5%, except in the case of one change per step where different in modularity increase to almost 25%.

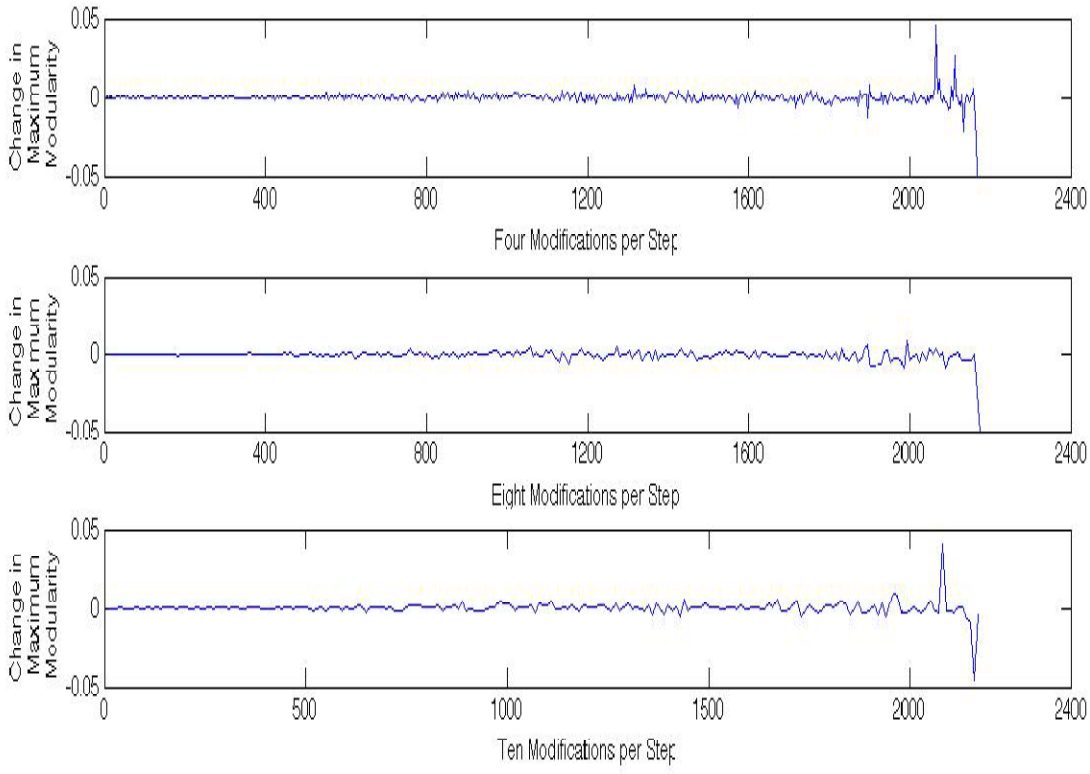


Figure 5.3: *Difference in maximum modularity of the static and dynamic method over each network. The X-axis plots the number of modifications and the Y-axis plots the difference in the modularity. Top: Four changes per time step, Middle: Eight changes per time step and Bottom: Ten changes per time step*

Then we compare the execution time of our dynamic algorithm with the original static algorithm. We compare the results only for time steps 1 to 2100, because solutions of the static and dynamic algorithms are equivalent in this range. We calculate the percentage of improvement using following formulae,

$$\frac{(StaticTime - DynamicTime)}{StaticTime} * 100$$

Figure 5.4, and 5.5 show that our dynamic algorithm is faster than the original static algorithm. Efficiency increases with the number of modifications. The speedup can be as much as 30% with an average of 13%.

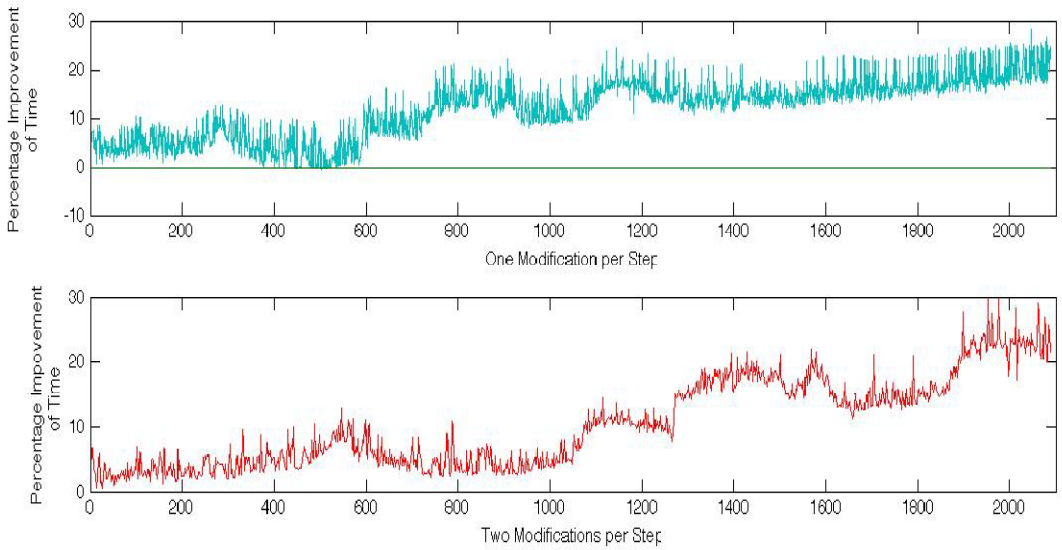


Figure 5.4: *Percentage speedup of the dynamic method over the static method at each network. The X-axis plots the number of modification and the Y-axis plots the speedup. Top: One change per time step and Bottom: Two changes per time step*

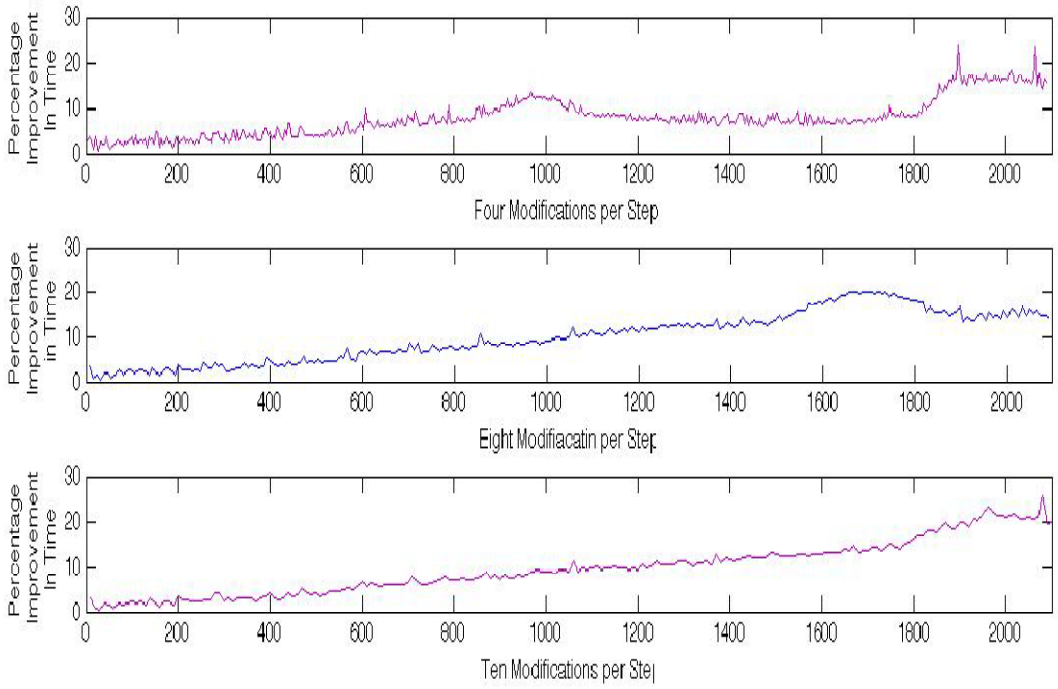


Figure 5.5: *Percentage speedup of the dynamic method over the static method at each network. The X-axis plots the number of modification and the Y-axis plots the speedup. Top: Four changes per time step, Middle: Eight changes per time step and Bottom: Ten changes per time step*

5.5 Discussion

Our goal has been to design an efficient algorithm for dynamic community detection by extending static agglomerative technique and comparing our results with the static algorithm results. We see from the results, that our algorithm improves the execution time of a static agglomerative method, while maintaining quality of solution as measured by the maximum modularity of the network. However, repeated applications of

the dynamic method too many changes of the same type can hamper the quality of the results.

Chapter 6

Conclusion and Future Work

Combinatorial properties can be useful in analysis of different types of networks. We used these properties to analysis of networks of two different areas; software engineering and bioinformatics. In software engineering, we quantify and evaluate the difference between networks representing different versions of JHotDraw 5. These graph theory based metrics provide important insight into understanding how JHotDraw evolved. This approach can be applied to understand the evolution of most complex networks systems. In bioinformatics, we used combinatorial properties to develop a new biclustering algorithm. Our primary results show that this approach looks promising to find different types of biclusters by comparing the similarity between networks. In final part, we designed an efficient community detection algorithm for dynamic networks by extending a static agglomerative method. Our dynamic algorithm can improve the execution time of a static agglomerative method.

As a part of future work, we can try to look into other metrics for large scale networks and algorithmic approaches for quantifying the disruption caused by large scale changes between versions of software networks and for finding the similarities between biological networks. Here, we demonstrated use of graph based metrics to evaluate the difference between networks representing versions of software system. However, this approach can be used to estimate the degree of change in evolving networks. In community detection, we can design dynamic community detection algorithm for

divisive methods. We can also improve the efficiency of the algorithm by a more selective search of the dendrogram.

References

[1] M. Newamn, *"Scientific collaboration networks: Ii. shortest paths, weighted networks, and centrality"*, Phys. Rev. E, vol. 016132, 2001.

[2] A. Barbour and D. Mollison, *"Stochastic Processes in Epidemic Theory"*. Springer, 1990, ch. Epidemics and random graphs, pp. 86–89

[3] J. L. Gross and J. Yellen, *"Handbook of Graph Theory and Applications"*. CRC Press, 2004.

[4] A. L. Barabasi, H. Jeong, Z. Neda, E. Ravasz A. Schubert, and T. Vicsek, *"Evolution of the social network of scientific collaborations", Phys*, vol. 311, pp. 590–614, 2002.

[5] C. R. Myers, *"Software systems as complex networks: Structure, function and evolvability of software collaboration graphs",* Phys. Rev. E, vol. 68, 2003.

[6] A. Chatzigeorgiou, N. Tsantalis, and G. Stephanides, *"Application of graph theory to OO software engineering",* in Proc. of the Intl. Workshop on Interdisciplinary Software Engineering Research (WISER '06), 2006, pp. 29–36.

[7] L. Wang, Z. Wang, C. Yang, L. Zhang, and Q. Ye, *"Linux kernels as complex networks: A novel method to study evolution",* in Intl. Conference on Software Maintenance (ICSM '09), 2009, pp. 41–50.

[8] M. Savi´c, M. Ivanovi´c, and M. Radovanovi´c, *"Characteristics of class collaboration networks in large Java software projects",* Information Technology and Control, vol. 40, no. 1, pp. 48–58, 2011.

[9] Brown P, Botstein D: *"Exploring the new world of the genome with DNA microarrays"*. Nature Genetics 1999, 21:33-37.

[10] Y. Cheng and G.M. Church, *"Biclustering of Gene-Expression Data",* Proc. Eighth Int'l Conf. Intelligent Systems for Molecular Biology (ISMB '00), pp. 93-103, 2000.

[11] J.A. Hartigan, *"Direct Clustering of a Data Matrix",* J. Am. Statistical Assoc. (JASA), vol. 67, no. 337, pp. 123-129, 1972.

[12] B. Mirkin, "Nonconvex Optimization and its Applications", Math. Classification and Clustering, Kluwer Academic Publishers, 1996.

[13] Harpaz R, Haralick R: "*Mining Subspace Correlations"*. IEEE Symposium on Computational Intelligence and Data Mining 2007, 335-342

[14] Dhillon I, Mallela S, Modha D: *"Information-theoretic co-clustering"*. Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining ACM Press New York, NY, USA; 2003, 89-98

[15] R. Peeters, *"The Maximum Edge Biclique Problem Is NP Complete",* Discrete Applied Math., vol. 131, no. 3, pp. 651-654, 2003.

[16] Madeira S, Oliveira A: *"Biclustering Algorithms for Biological Data Analysis: A Survey"*. IEEE Transactions on Computational Biology and Bioinformatics 2004, 1:24-45.

[17] Tanay A, Sharan R, Shamir R: *"Biclustering Algorithms: A Survey"*. Handbook of Computational Molecular Biology 2005, 9:26-1.

[18] D. Bozdag, A.S. Kumar, U. Catalyurek: *"Comparative Analysis of Biclustering Algorithms",* ACM International Conference on Bioinformatics and Computational Biology, 2010

[19] T. Murali and S. Kasif, *"Extracting Conserved Gene Expression Motifs from Gene Expression Data",* Proc. Pacific Symp. Biocomputing, vol. 8, pp. 77-88, 2003.

[20] A. Tanay, R. Sharan, and R. Shamir, *"Discovering Statistically Significant Biclusters in Gene Expression Data",* Bioinformatics, vol. 18, pp. 36-44, 2002.

[21] J. Ihmels, S. Bergmann, and N. Barkai, *"Defining Transcription Modules Using Large-Scale Gene Expression Data",* Bioinformatics, vol. 20, pp. 1993-2003, 2004.

[22] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini, *"Discovering Local Structure in Gene Expression Data: The Order-Preserving Sub- Matrix Problem",* Proc. Sixth Ann. Int'l Conf. Computational Biology (RECOMB '02), pp. 49-57, 2002.

[23] D. Bozdag, J. D. Parvin, and U. Catalyurek. *"A biclustering method to discover co-regulated genes using diverse gene expression datasets"*. In Proc. of International Conference on Bioinformatics and Computational Biology, volume 5462, pages 151–163, April 2009.

[24] Prelic, A., et al., *"A systematic comparison and evaluation of biclustering methods for gene expression data",* Bioinformatics 2006 22(9):1122-1129.

[25] M. Porter, N. Mucha, P. J., M. E. J., A. J. Friend, *"Community structure in the united states house of representatives",* Physica A. 386 (2007) 414–438.

[26] A. L. Barabasi, H. Jeong, E. Ravasz, Z. Neda, A. Schuberts, T. Vicsek, *"Evolution of the social network of scientific collaborations",* Physica A. 311 (2002) 590–614.

[27] K. Voevodski, S. H. Teng, Y. Xia, *"Finding local communities in protein networks"*, BMC Bioinformatics 10 (10) (2009) 297

[28] K. Atkins, J. Chen, V. S. Anil Kumar, A. Marathe, *"Structure of electrical networks: A graph theory based analysis",* International Journal of Critical Infrastructures 5 (2009) 265–284.

[29] M. E. J. Newman, M. Girvan, *"Finding and evaluating community structure in networks",* Phys. Rev. E 69 (2) (2004) 026113.

[30] A. Clauset, M. E. J. Newman, C. Moore, *"Finding community structure in very large networks",* Phys. Rev. E 70 (6) (2004) 066111.

[31] Guimera R, Sales M and Amaral, *"Modularity from fluctuations in random graphs and complex networks",* 2004 *Phys. Rev.* E 70 025101

[32] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre, *"Fast unfolding of communities in large networks",* Journal of Statistical Mechanics: Theory and Experiment, 1742-5468, P10008 (12 pp.), 2008

[33] C. Tantipathananandh, T. Berger-Wolf, D. Kempe, *"A framework for community identification in dynamic social networks",* in: Proceedings of the 13th ACM

SIGKDD international conference on knowledge discovery and data mining, 2007, pp. 717–726.

[34] H. Ning, W. Xu, Y. Chi, Y. Gong, T. Huang, *"Incremental spectral clustering with application to monitoring of evolving blog communities",* in: SIAM Int. Conf. on Data Mining, 2007, pp. 261–72.

[35] I. X. Y. Leung, P. Hui, P. Li`o, J. Crowcroft, *"Towards real-time community detection in large networks",* Phys. Rev. E 79 (2009) 066107.

[36] P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, J.-P. Onnela, *"Community structure in time-dependent, multiscale, and multiplex networks",* Science 328 (2010) 876–878.

[37] E. B. Swanson, "The dimensions of maintenance", in Intl. Conference on Software Engineering (ICSE '76), 1976, pp. 492–497.

[38] I. Thapa and H. Siy, "Assessing the impact of refactoring activities on the JHotDraw project", in ACM Symp. on Applied Computing, 2010.

[39] JHotDraw, http://www.jhotdraw.org [15 September 2010]

[40] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto, *"Ranking significance of software components based on use relations",* IEEE Transactions on Software Engineering, vol. 31, no. 3, pp. 213–225, March 2005

[41] R. Yokomori, H. Siy, M. Noro, and K. Inoue, *"Assessing the impact of framework changes using component ranking",* in *Intl. Conference on Software Maintenance (ICSM '09)*, Edmonton, Canada, 2009, pp. 189–198

[42] Matlb BGL Library, http://www.stanford.edu/˜dgleich/programs/matlab bgl/

[43] P. Paymal, R. Patil, S. Bhowmick and H. Siy, *"Measuring Disruption From Software Evolution Activities Using Graph-Based Metrics*", Proceedings of the International Conference on Software Maintenance (ICSM) (ERA Track) 2011

[44] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, *"Graphviz and dynagraph - static and dynamic graph drawing tools",* in *Graph Drawing Software*. Springer-Verlag, 2003, pp. 127–148

[45] B. H. Good, Y. de Montjoye, and A. Clauset, *"The performance of modularity maximization in practical contexts", Phys*, vol. 82, p. 046106, 2010.

[46] McQuitty LL: *"Similarity analysis by reciprocal pairs for discrete and continuous data."* Educational Psychol Measurement 1966, 26:825-831.

[47] MacQueen J: *"Some methods for classification and analysis of multivariate observations"*. In Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability. Statistics. Edited by: Le Cam LM, Neyman J. Berkeley, CA, USA: University of California Press; 1967:1:281-297

[48] K. Y. Yip, D. W. Cheung, and M. K. Ng. Harp: *"A practical projected clustering algorithm"*. IEEE Transactions on Knowledge and Data Engineering, 16(11):1387–1397, 2004.

[49] H. Cho, I. Dhillon, Y. Guan, and S. Sra., *"Minimum sum-squared residue based co-clustering of gene expression data"*. In SIAM International Conference on Data Mining, pages 114–125, 2004.

[50] O. Kuchaiev, T. Milenkovic, V. Memisevic, W. Hayes, N. Przulj *"Topological network alignment uncovers biological function and phylogeny", Journal* of the Royal Society Interface, 2010, 7:1341-1354

[51] Milenkovic´, T. & Prz˘ulj, N. 2008, *"Uncovering biological network function via graphlet degree signatures"*. Cancer Inform. 6, 257−273.

[52] J. Handl, J. Knowles, and D. B. Kell, *"Computational cluster validation in post-genomic data analysis",* Bioinformatics 21 (15), 3201, Year 2005.

[53] I. Gat-Viks, R. Sharan, and R. Shamir, *"Scoring clustering solutions by their biological relevance",* Bioinformatics, 19: 2381-2389

[54] A. Prelic, S. Bleuler, Philip, Zimmermann, A. Wille, P. Bühlmann, W. Gruissem, L. Hennig, L. Thiele, E. Zitzler, *"A systematic comparison and evaluation of biclustering methods for gene expression data".* Bioinformatics, 2006: 1122~1129

[55] Barkow, S., Bleuler, S., Prelic, A., Zimmermann, P., and E. Zitzler. *"BicAT: a biclustering analysis toolbox,"* Bioinformatics, 2006 22(10):1282-1283

[56] Jaccard, Paul (1901), *"Étude comparative de la distribution florale dans une portion des Alpes et des Jura"*, Bulletin de la Société Vaudoise des Sciences Naturelles **37**: 547–579

[57] Elena Deza & Michel Deza, "Encyclopedia of Distances", page 94, Springer

[58] K. Wakita, T. Tsurumi, *"Finding community structure in mega-scale social networks",* in: Proceedings of the 16th international conference on World Wide Web, ACM, 2007, pp. 1275–76

[59] S. Bansal, S. Bhowmick and P. Paymal, *"Fast Community Detection For Dynamic Complex Networks",* Communications in Computer and Information Science Volume 116. Proceedings of the Second Workshop on Complex Networks, CompleNet 2010

[60] D. A. Bader, A. Amos-Binks, C. Chavarrsa-Miranda, D. andHastings, K. Madduri, P. S. C., *"STINGER: Spatio-Temporal Interaction Networks and Graphs (STING) Extensible Representation",* Tech. rep., Georgia Institute of Technology (2009).

[61] Y. Saad, *"Iterative Methods for Sparse Linear Systems",* PWS Publishing Company, 1995

[62] The DBLP Computer Science Bibliography, http://dblpVis.uni-trier.de