8-2017

Detection of Comparability Subgraphs from Large Networks

Muthunagai Balakrishnamoorthy
University of Nebraska at Omaha

Detection of Comparability Subgraphs from Large Networks

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

Masters of Science

University of Nebraska at Omaha

By

Muthunagai Balakrishnamoorthy

August, 2017

Supervisory Committee
Chair Dr. Sanjukta Bhowmick
Dr. Hesham Ali
Dr. Kathryn Cooper

Detection of Comparability subgraphs from Large Networks

Muthunagai Balakrishnamoorthy, MS

University of Nebraska, 2017

Advisor: Dr. Sanjukta Bhowmick

Abstract

Real world large scale networks can be represented as graphs. This approach plays a key role in analysis in the domains of social networks [1] and bioinformatics [2], among others. Analyzing these networks is computationally complex and expensive, especially in terms of memory and time complexity. A popular technique subverting time and computation expense for analyzing networks is extracting substructures, which preserves more important information and less noise [12]. In this work, we use special a special substructure called comparability, which preserves transitive orientation. Our motive is to extract a maximal comparability subgraph since no algorithm exists. Our algorithm is able to find a maximal comparability subgraph from both undirected and directed graphs. Finding a clique of given size is a NP-complete problem, so we must implement some additional constraints to maximize time efficiency. If the given input graph is chordal, then extraction of the clique of size n becomes a problem that is solvable in polynomial time. So we have written an algorithm to find the clique of given size, and implemented the algorithm to find a maximal chordal subgraph. Since we worked on two different special subgraphs, we compared our results to investigate whether the given graph is chordal or comparability in nature. In our research, we have proposed a parallel sampling method for efficient network analysis.

Acknowledgements

It is a great pleasure for me that I have got an opportunity to acknowledge the support of my advisor Dr. Sanjukta Bhowmick. She offered me a great assistance, guidance and lots of support throughout my thesis. She has been an encouraging mentor and offered me a great chance to develop myself towards the positive approach and self confidence. I am very thankful to Dr. Hesham Ali and Dr. Kathryn Cooper for the invaluable assistance in this thesis. I also want to thank Ishwor Thapa for his great help in analyzing the biological networks.

I really want to thank my husband for his support and motivation throughout my studies and especially in this thesis period. I want to thank my little son for sacrificing his lot of mom & son's together time towards this thesis & graduate course success. I would also love to thank my family members and friends who always encouraged & supported me a lot.

Table of Contents

1. Introduction **1**

2. Background **5**

2.1 Graph Theory Terminologies: **5**

2.2 Graph Filtering **7**

3. Methodology **8**

3.1 Comparability Graphs **8**

3.1.1 Data structure 9

3.1.2 Comparability Algorithm for Undirected graphs: 10

3.1.3 Comparability Algorithm for directed graphs: 14

3.1.4 Parallel pattern for Comparability subgraph 20

3.1.5 Analysis of different ordering impact on Comparability filter 21

3.1.6 Comparability nature analysis over PPI network 23

3.2 Chordal Graphs **28**

3.2.1 Algorithm to find Maximal Chordal Graph 28

3.2.2 Maximal Chordal Subgraphs of Social Networks 29

3.2.3 Algorithm to find a clique of size n 30

4.Experimental Results **32**

4.1 Clique size vs No of cliques **32**

4.2 Comparison between chordal and comparability **40**

Conclusion & Future work **42**

List of Figures

Figure 2.1 Representation of Sample Undirected graph 5

Figure 3.2 Array of linked list representation of edges 9

Figure 3.2 Comparability graph of an Undirected graph 11

Figure 4.1 Clique size vs number of cliques. 32

Figure 4.2 Comparison between chordal and comparability 40

Figure 4.3 Comparison graph between different subgraphs. 41

Chapter 1

1. Introduction

Networks can be described as an interrelated or interacting group of objects. In these networks (also known as graphs), objects are represented as nodes and the interrelation between the objects is represented as edges. Chemical structures, protein structures, computer networks, telephone networks and social networks are some of the real world examples of networks. Network analysis is used to compute structural properties such as who is connected to whom. This helps us to enhance the network quality. For example, Facebook has more than 1 billion active users [13]. Though it is a very large scale network, Facebook is able to find related properties between users and suggesting friends, movies, books etc., based on a user's interest. Network analysis is one of the key techniques in analyzing social networks.

The purpose of network analysis is to extract a special structure from large scale networks. Analyzing these networks is a cumbersome process since the networks are really complex and extensively big. Storing and analyzing a large scale network is a complicated process in terms of memory and computation. There are two popular methodologies to handle the analysis of a large scale network (1) extracting special substructures which contain certain graph theoretic properties (2) using high performance super computers with multiple processors [3,4,5]. In a large network, some edges are important and some are not important. To reduce the complexity of network, substructures can be extracted with more information (important edges) and less noise

(unimportant edges). Filtering allows us to analyze smaller, easier to see substructures and apply those conclusions to the original network. So by using a filtering method, we are able to extract a substructure of the original graph with its own characteristics [8,9,10].

The motive of this thesis is to extract a special substructure from a given network. There are two special substructures called comparability and chordal. The first one is Comparability Subgraph which is a directed subgraph of a given graph which is transitive in nature. Many relations in our real life are transitive in nature. Transitive relations can be defined as follows. If Bob knows Albert and Albert knows James, then Bob may know James. In terms of biological networks, the transitive relation can be useful to find driver cells. For example, normally cell C can be reached by moving from A to B and B to C, if we know that the structure is transitive, then we can reach cell C directly from cell A. The second substructure is Chordal Subgraph. Chordal graphs are perfect graphs which maintain triangular structure. That is, all the cycles with size 4 or more will have a chord, which is not part of that cycle, that connects two vertices of the cycle.

From a large network, we can extract any number of substructures with different sizes. But the larger subgraph is called maximal subgraph. Maximal subgraph can be defined as the larger subgraph in which we can not add at least one more edge to the resultant subgraph. Currently there is no existing algorithm to extract the maximal comparability subgraph. So we wrote an algorithm to find maximal comparability subgraph. We have also proposed a parallel template for comparability graph which can

be used in high performance computers. In this parallel template, the network data can be divided into partitions and the computation occurs in different processors at a same time. It helps to reduce the time complexity in network analysis.

Both substructures can be extracted from a given undirected graph. Since we are able to extract two different maximal subgraphs, we are able to compare the results to figure out whether the network is closer to either comparability or chordal in nature. We have also compared the results of two different substructures. Though the comparability subgraph is computed from an undirected graph, we have created an algorithm to find maximal comparability subgraph from given directed graph. We used biological pathways to test this algorithm since they are directed. Finding a clique of given size is an NP-complete problem. We created an algorithm to find a clique of given size from a given chordal graph in polynomial time. We have extended our research to investigate how different random ordering affects the result. By randomly ordering the vertices of the graph, we were able to get different maximal comparability subgraphs. We have also applied our comparability filter on Protein Protein Interaction network.

Outline of Thesis:

This Thesis Report is organized as follows. In Chapter 2, we have given some background of graph theory concepts. In Chapter 3, we have presented our newly developed algorithm and implementation explanation to find comparability subgraph and how it works for directed. We have discussed the algorithm to find maximal chordal graph and how to find the clique of a given size from given chordal graph. In Chapter 4, we have derived the experimental results and analysis which includes comparison of original graph and subgraph properties. Also we have presented the results of the comparison between the chordal and comparability subgraph properties. In Chapter 5, we presented our concluding remarks and future work on further research.

Chapter 2

2. Background

Graphs can be described as the graphical representation or model of social networks. Many real world structures can be represented as graphs. Nodes of a network can be represented as vertices and communication between those nodes can be represented as edges. A graph is a collection of vertices and edges in which the edges connect pair of vertices. Graphs are represented graphically by drawing a dot or circle for each vertex and arc/line between two vertices to represent an edge. The communication direction between the edges can be represented using arrows. The graphs can be either directed or undirected in nature.

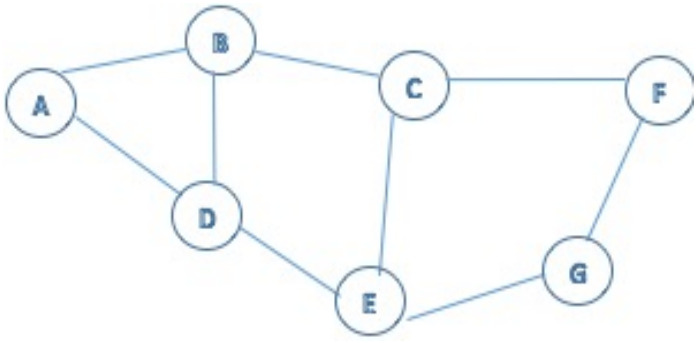


Figure 2.1 Representation of Sample Undirected graph

2.1 Graph Theory Terminology:

We introduce some graph terminology that will be helpful to understand the explanation of algorithms [6].

Graph:

The graph can be represented as G=(V, E). Here V is set of vertices and E is set of edges.

Vertices and Edges:

An edge e € E is connecting two vertices u and v, which are called its endpoint vertices. A vertex u is said to be a neighbor of vertex v, if they are connected by an edge. In Figure 2.1, there are a total 7 vertices and 9 edges in the graph.

Cycle:

A path is an alternating sequence of vertices and edges. A cycle is a path where the initial and end vertices are identical. In Figure 2.1, vertices (B, C, E, D, B) form a cycle. This cycle started with B and ended at B.

Clique:

A clique is a set of vertices in which all the vertices are connected to each other. In Figure 2.1, vertices (A, B, D) forms a clique because everyone in the group is connected to each other.

Degree:

Degree of a vertex can be defined as the number of connections it has with other vertices. The Degree of vertex v is denoted as Deg(v). Vertices with high degrees are called hub vertex. In Figure 2.1, Degree of vertices are, Deg(A) = 2, Deg(B) = 4, Deg(C) = 4, Deg(D) = 2, Deg(E) = 4, Deg(F) = 2 and Deg(G) = 2.

Clustering Coefficient:

Clustering Coefficient is a measure, which describes the proportion of acquaintances of a vertex with its neighbors. In Figure 2.2, Clustering Coefficients values of vertex CC(A) = 1, vertex CC(B) = 0.3, vertex CC(C) = 0, vertex CC(D) = 0.3, vertex

CC(E) = 0, vertex CC(F) = 0, vertex CC(G) = 0.

Perfect graphs:

A perfect graph is a graph in which the size of the largest clique of the graph is equal to the chromatic number of the graph. Comparability and Chordal graphs are part of the perfect graph family.

Comparability graph:

Comparability graph is an undirected graph that connects pairs of elements that are comparable to each other in partial order.

Chordal graph:

Chordal graph is a perfect graph, in which every induced cycle in the graph should have exactly three vertices.

2.2 Graph Filtering

Graph filtering can be achieved by retaining some of the structural or functional properties of the original graph. There are many types are filtering that can be defined by filtering based on many graph theoretic properties. Chordal filtering is one of the types which is popular among the researchers in which the filter retains the triangular structures of the original graph. Comparability filtering is an another type of filter which retains the transitive nature from the original graph.

Chapter 3

3. Methodology

Extraction of special substructure helps to improve the analysis of large scale network in terms of memory and cost. The filtering method is effective in extracting the subgraph from the original network with structural properties. There are a number of algorithms that are being used to extract a subgraph with a specific structure. Any maximal subgraph contains the maximal number of edges from the original network which maintains a particular structure and graph theoretical property. The largest possible subgraph of the given graph is called maximal subgraph in which we can not find another edge anywhere in the graph such that it could be added to the subgraph and all the edges in the subgraph would still preserve its intended structure or property. The main objective of our research is to find a maximal comparability subgraph and maximal chordal subgraph.

3.1 Comparability Graphs

As discussed in the background section, comparability graph is a subgraph of a given undirected graph, in which the transitive property is maintained throughout the graph by applying the directions to the edges and verifying for the transitive property. If A is connected to B and B is connected to C then, A should be connected to C. Comparability graphs are perfect graphs. Many hard problems such as graph coloring and independent set problem can be calculated in polynomial time when the input graph is a

comparability graph. In this thesis, we have created an algorithm to find a maximal comparability subgraph from the given graph.

3.1.1 Data structure

In our algorithm, we have taken an undirected graph as an input, in which the input data is given as a list of edges and each edge is represented by its source and target vertices. After reading this input edges from the file, we are creating the array of linked lists to store the neighbor relationship of a vertex. So the array of linked list can be represented in picture as below.

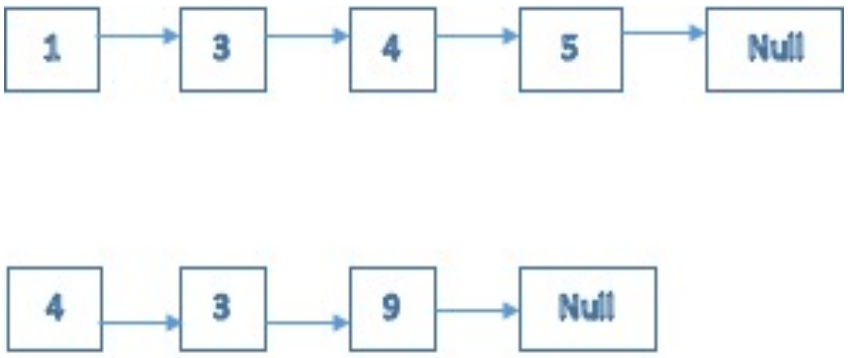
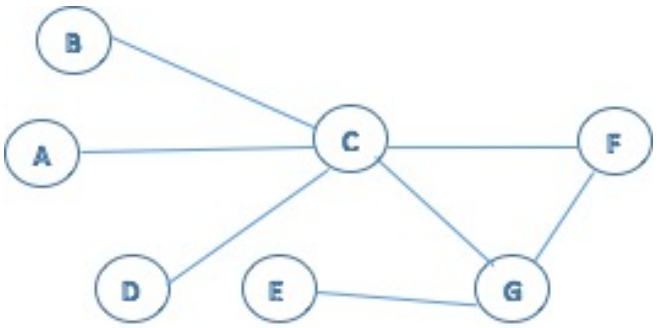


Figure 3.2 Array of linked list representation of edges

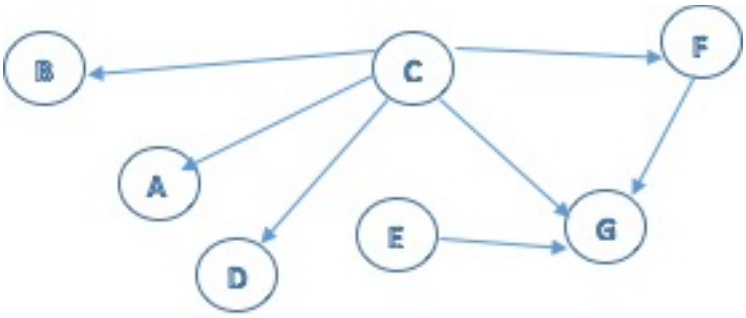
Here the starting node 1 is the vertex and 3, 4 and 5 are the neighbors of vertex 1. So the given input edges are 1->3, 1->4, 1->5. Since the given graph is an undirected in nature, while adding the neighbors to vertex 1, we will add vertex 1 as the neighbor of 3, 4 and 5.

3.1.2 Comparability Algorithm for Undirected graphs:

The objective of our algorithm is to maintain the triangular structure in subgraphs with strict transitive order. Our comparability subgraph algorithm is a sequential method of deriving the maximal comparability subgraph. Here we start with the vertex and growing by adding edges who are maintaining the transitive property after applying directions. Initially we assume all the vertices are sink. Take all the vertices from the input graph G and add them to the resultant graph G', which doesn't have any edges when we start. By looping through the vertex one by one, we look for the neighbors of that particular vertex and add them one by one to the resultant graph G', if it is maintaining the transitive property. If transitivity is not satisfied, then do not add that edge to the resultant graph G'. Our comparability subgraph algorithm preserves the transitive property throughout the graph and assumes the given input graph is a connected graph.



(a)



(b)

Vertex	A	B	C	D	E	F	G
Type	Sink	Sink	Source	Sink	Source	Neutral	Sink

(c)

Figure 3.2 Comparability graph of an Undirected graph

a) Undirected Input Graph b) Output Comparability Subgraph c) Vertices and its type in Comparability Subgraph

Algorithm: To find comparability subgraph from given undirected graph.

Input G(V, E) : V- vertex set and E – edge set

Output G'(V, E) – Comparability subgraph of G.

Create a graph G'(V,E) as E is empty and G'(V)=G(V)

Create a queue Q

Initialize all the vertices as sink in G'

Start with the vertex Vs

Add Vs to Q

While Q is not empty

Set v as first element in Q

```
For all neighbor u of v
    If v is sink
        transity = Call Checktransitivity(v,u,u→v)
        If transit is yes
            Add u→v to G'
            If u is empty & sink
                u=source
            else if u is !empty & sink
                u=neutral
            else if u is !empty & source
                u=source
            push u to Q.

    If v is source
        transity = Call Checktransitivity(v,u,v→u)
        If transit is yes
            Add v→u to G'
            If u is empty & sink
                u=sink
            else if u is !empty & sink
                u=sink
            else if u is !empty & source
                u=neutral
```

push u to Q.

If v is neutral

transity = Call Checktransitivity(v,u,v→u)

If transit is yes

Add u→v to G'

Checktransitivity(u,v,edge)

If v is sink & u is sink

Return yes

Else if v is source & u is source

Return yes

Else if v is sink & u is source

If edge is u→v

Return yes

If edge is v→u

Return no

Else if v is source & u is sink

If edge is v→u

Return yes

If edge is u→v

Return no

Else if v is sink & u is neutral

If all neighbors of u, v are common & u has extra one neighbor

If edge is u→v

Return yes

If edge is v→u

Return No

Else if v is source & u is neutral

If all neighbors of u, v are common & u has extra one neighbor

If edge is v→u

Return yes

If edge is u→v

Return no

Else if u is neutral & v is neutral

if all neighbors of u, v are common

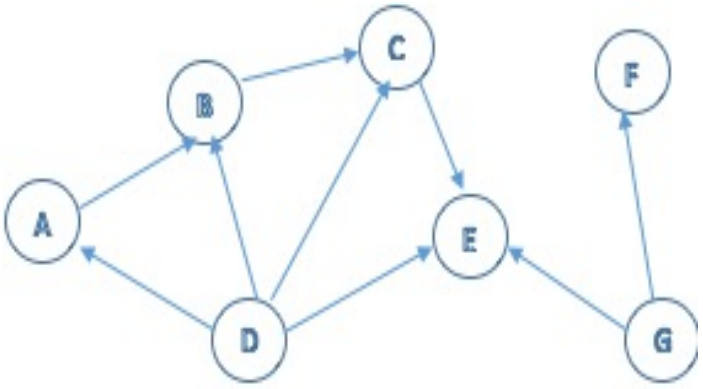
Return yes

Else

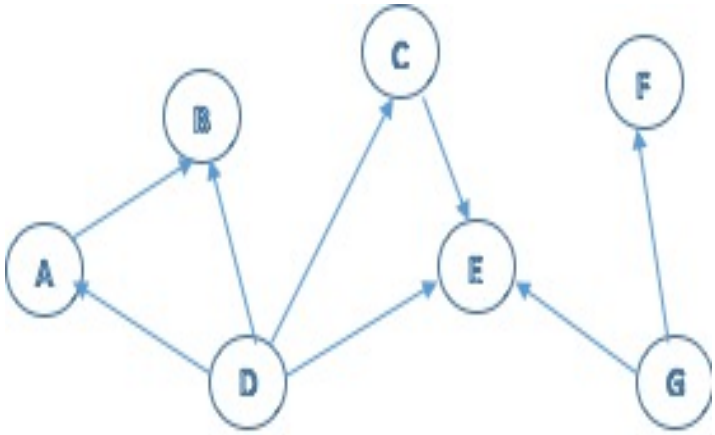
Return no

3.1.3 Comparability Algorithm for directed graphs:

Though the comparability subgraphs are derived from undirected graphs by applying directions to maintain transitivity, we have implemented an algorithm to find a comparability subgraph from the given directed graph. Here we are using the directed graph as an input and we are checking for the transitivity property for each given directed edge. If the transitivity property is maintained, then we are adding that edge to the resultant graph.



(a)



(b)

Vertex	A	B	C	D	E	F	G
Type	Neutral	Sink	Neutral	Source	Sink	Sink	Source

(c)

Figure 3 .3 Comparability subgraph for directed graphs

a) Directed Input Graph b) Output Comparability subgraph c) Vertices and its type in comparability subgraph

The algorithm steps to find a comparability subgraph from the given directed graph are described below:

Algorithm: To find comparability subgraph from given Directed graph.

Input graph G(V,E) : V- vertex set and E – edge set

Output graph G'(V,E) – Comparability subgraph of G.

Create a graph G'(V,E) as E is empty and G'(V)=G(V)

Initialize all the vertices as sink in G'

Start with the vertex Vs

For all the vertices in V

For all neighbor u of v

If v is sink

transity = Call Checktransitivity(v,u,v→u)

If transit is yes

Add v→u to G'

If v is sink & u is sink

v=source

else if v is sink & u is source

u=neutral & v = source

else if v is sink & u is neutral

v=source

if v is source

transity = call checktransitivity(v,u,v→u)

if transity is yes

```
                    Add v→u to G'
                    if v is source & u is sink
                        no change
                    else if v is source & u is source
                        u= neutral
                    else if v is source & u is neutral
                        no change.
            if v is neutral
                if u is sink
                transity = Call Checktransitivity(v,u,v→u)

                If transit is yes
                    Add v→u to G'
Checktransitivity(u, v, edge)
    If v is sink & u is sink
        If u and v has no neighbors and both are not neighbor of any vertex
            Return yes
        Else if u and v has no neighbors and v is not neighbor of any vertex
            Return yes
        Else if v is neighbor of an vertex and u is not
            Return no
        Else if v and u are the neighbor of an vertex
            Return yes
```

Else if v is source & u is source

If v and u has a common neighbor

Return yes

Else if v is sink & u is source

If v is neighbor of a vertex and u has neighbor

If v is a neighbor of a vertex and that vertex is neighbor of u

Return yes

Else

Return no

Else if v is not a neighbor of any vertex but u has neighbors

Return no

Else if v is source & u is sink

Return yes

Else if v is sink & u is neutral

If v is neighbor of a vertex and u has neighbor

If v is an neighbor of a vertex and that vertex is neighbor of u

Return yes

Else

Return no

Else if v is not a neighbor of any vertex

Return no

Else if v is source & u is neutral

If all neighbors of v and u has common neighbors

Return yes

Else

Return no

Else if v is neutral & u is neutral

if all neighbors of u, v are common

Return yes

Else

Return no

Else if v is neutral & u is sink

If u and v are common neighbor of vertices

Return yes

Else

Return no

Else if v is netural & u is source

If v and u has common neighbors

Return yes

Else

Return no

Else if v is neutral & u is neutral

If all the neighbors of v and u are common neighbors

Return yes

Else

Return no

3.1.4 Parallel pattern for Comparability subgraph

As we discussed, the real time large scale networks are really big. The technical implementation of our algorithm is effective in terms of accuracy but may not be effective when we use an extensively big network. So we propose a parallel pattern which improves the performance of an algorithm. The motive of our proposal is to reduce the processing time of the algorithm. Processing a very large scale network in a single processor may be a time consuming process. Our algorithm takes different timing to process a given input graph based on its size. A very large network with more than hundred and seventy thousand edges has been running for more than 240 minutes for ten iterations. This made us think about the parallel processing for our algorithm.

In this parallel pattern, the process can be divided among the high performance computational units and running them in parallel would help to reduce the total time to run the algorithm. We can divide a given input graph in to different partitions and use each partition as input to different processors and derive the comparability subgraph independently. We are able to list some of the edges which did not fall under these partitions and collect them in one place and use them to find out the edges which can be added later based on the type of the vertex, after finding the individual partition subgraphs. We can add some of these missed edges while partitioning to the result, whose end point vertex pair has a type either v(source) → u(sink) or v(sink) ← u(source).

3.1.5 Analysis of different ordering impact on Comparability filter

We have extended our result to compare the results of the comparability filter based on the different breath first search order. BFS algorithm is used for traversing a graph. It starts from a root vertex and find all the neighbors before moving to the next level of neighbor vertices. BFS ordering can be defined as the enumeration of the vertices of the given graph. To achieve the different breath first search ordering for the same graph, we have randomly reordered(renamed) the vertices and marked its neighbors accordingly how they are present in the original graph. We have tried up to 50 shuffles for smaller networks and found most of them were giving the similar count as result. We have stored the size of the outputs in an array and taken the largest one among the subgraphs we obtained. When the comparability subgraphs are similar in size, we choose one as our output which has lesser processing time for our comparison process. To avoid the manual error in the selection, we have stored in an array and found the bigger subgraph by sorting the array. The below picture represents the difference in the filter for different vertex ordering. The below table represents the time taken for each iteration to shuffle the edges and extract the comparability subgraph with mentioned number of edges retained.

shuffle	No of Edges Retained	Time(ms)
1	54	1.36027
2	54	1.1057
3	54	0.6951
4	54	0.681156
5	54	0.895766
6	54	0.758115
7	54	0.694511
8	54	0.847753

9	54	0.646082
10	54	0.649605
11	54	0.777931
12	54	0.611494
13	54	0.724716
14	54	1.05429
15	54	0.712628
16	54	0.734787
17	54	0.749268
18	54	0.647886
19	54	0.783512
20	35	0.68468
21	35	0.621295
22	35	1.20501
23	35	1.12763
24	35	0.739215
25	35	0.592884
26	35	0.53356
27	35	0.898151
28	35	0.596496
29	35	0.884333
30	35	0.847442
31	35	0.787016
32	35	0.920569
33	35	0.947185
34	35	0.789104
35	35	0.791999
36	35	1.4558
37	35	0.850566
38	35	0.943371
39	35	0.864129
40	35	1.51071
41	42	0.578792
42	42	0.52563
43	42	0.682146
44	42	0.669366
45	42	0.66224
46	42	0.690379
47	42	0.605454

48	42	0.616778
49	42	0.610603
50	42	0.759286

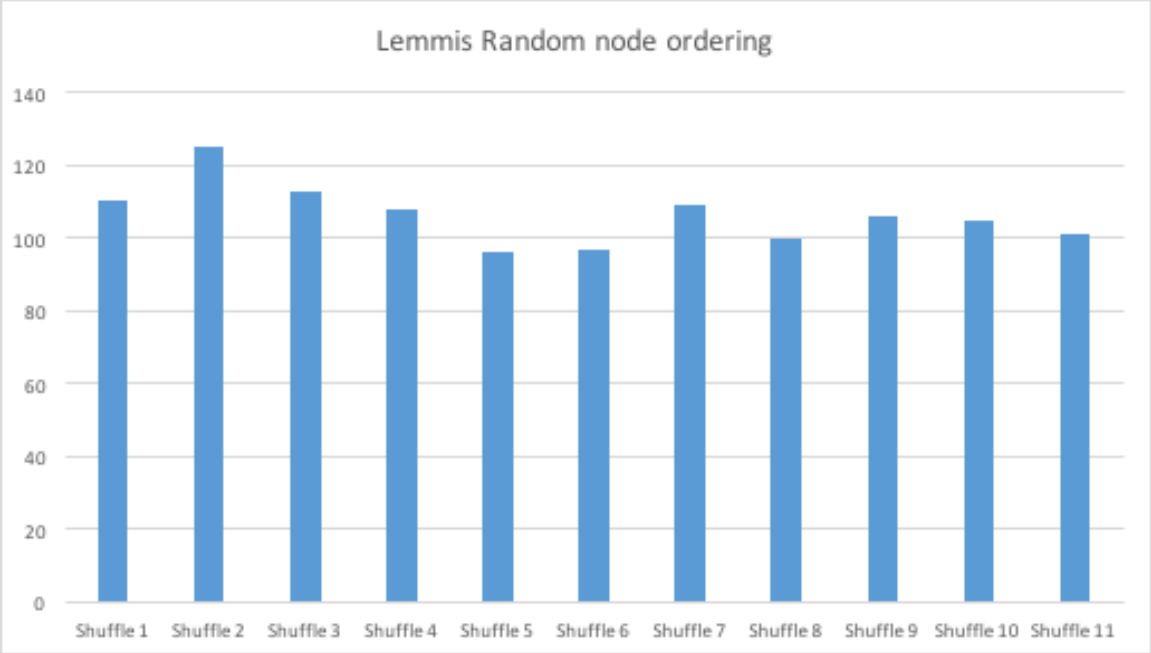


Figure 3.4 BFS Ordering Vs Comparability filter results

Graph	Shuffle 1	Shuffle 2	Shuffle 3	Shuffle 4	Shuffle 5	Shuffle 6	Shuffle 7	Shuffle 8	Shuffle 9	Shuffle10
Adjnoun	192	167	211	162	182	170	177	160	188	180
asin	13712	14024	12711	14310	13810	13613	13981	13215	14339	14409
Karate	54	54	54	35	54	42	35	42	42	54
Lemmis	110	125	113	108	96	97	109	100	106	105
netscience	1696	1664	1710	1649	1690	1707	1688	1673	1665	1700

Figure 3.5 BFS Ordering Vs Comparability filter results on social networks

3.1.6 Comparability filter analysis over PPI network

We have downloaded the data from the data source http://mips.helmholtz-muenchen.de/corum/#download which gives the proteins to complex information. In other words, it gives the name of the proteins in every complex. We have applied the comparability filter on this network and extracted the comparability subgraph. Now we have applied the clustering in cytoscape using http://apps.cytoscape.org/apps/pewcc on both original and filtered protein protein interaction network. Then we have calculated the jaccard index. The jaccard index can be defined as the intersection over union. It is used to compare similarity and diversity of the sampled sets. In our analysis, we have identified the number of proteins in a complex and the number of proteins in a cluster and the jaccard index value for all the complexes. Based on the jaccard index value, we were able to understand the few similarity as mentioned below between original and comparability subgraph.

complex	Filtered jaccard index	complex jaccard index
TFIIIC containing complex	1	1
GINS complex	1	1

Figure 3.6 Complex High Similarity between original & comparability graph

Also we were able to see some of the complexes in comparability subgraph are having high value than the value computed from the original graph as below.

complex	Filtered jaccard index	complex jaccard index
Multisynthetase complex	1	0.9167
TAK1 complex	1	0.5
Cleavage stimulation factor	1	0.75

Figure 3.7 Comparability graph advantage complex

Similarly, it is possible to find clusters in filtered network that was difficult to find from the original network though they exist since our filtered network is a subset of the original network. This is because of the heuristic nature of finding clusters from large networks. Using this filtered smaller networks helps in finding clusters easier. below listed complexes are having clusters on the filtered subgraph.

HDAC1-associated core complex cI
CAV1-VDAC1-ESR1 complex
GammaH2AFX-NDHII-Ku70-DNA complex
SMN complex (GEMIN6,7, UNRIP), SMN-independent intermediate
Homodimeric complex LTBR
LINC complex, quiescent cells
Polycomb repressive complex
CTLH complex
Homodimeric complex LTBR
MSH2-MSH6 complex
POSH-AKT2 complex
BRMS1-RBP1 complex
p400-associated complex
BRD4 complex
CAND1-CUL2-RBX1 complex
CAND1-CUL2-RBX1 complex
CAND1-CUL2-RBX1 complex
CAND1-CUL2-RBX1 complex
CDC2-PCNA-CCNB1-GADD45G complex
CDC2-PCNA-CCNB1-GADD45G complex

Figure 3.8 Clusters part of filtered graph (not part of co-expression)

On the other hand, below listed complexes are present in both original and filtered network with the same value.

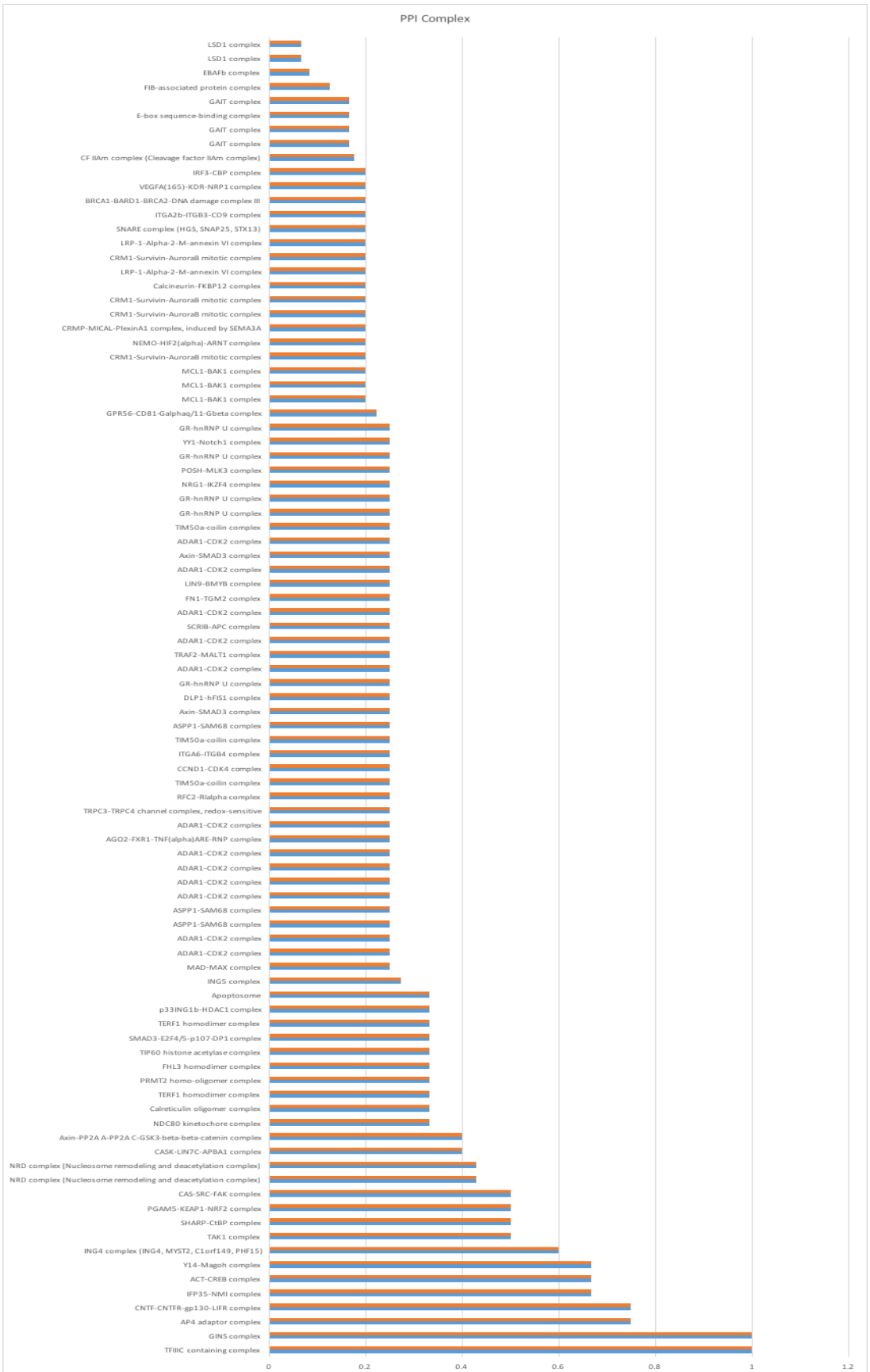

PPI Complex
LSD1 complex
LSD1 complex
EBAFb complex
FIB-associated protein complex
GAIT complex
E-box sequence-binding complex
GAIT complex
GAIT complex
CF IIAm complex (Cleavage factor IIAm complex)
IRF3-CBP complex
VEGFA(165)-KDR-NRP1 complex
BRCA1-BARD1-BRCA2-DNA damage complex III
ITGA2b-ITGB3-CD9 complex
SNARE complex (HGS, SNAP25, STX13)
LRP-1-Alpha-2-M-annexin VI complex
CRM1-Survivin-AuroraB mitotic complex
LRP-1-Alpha-2-M-annexin VI complex
Calcineurin-FKBP12 complex
CRM1-Survivin-AuroraB mitotic complex
CRM1-Survivin-AuroraB mitotic complex
CRMP-MICAL-PlexinA1 complex, induced by SEMA3A
NEMO-HIF2(alpha)-ARNT complex
CRM1-Survivin-AuroraB mitotic complex
MCL1-BAK1 complex
MCL1-BAK1 complex
MCL1-BAK1 complex
GPR56-CD81-Galphaq/11-Gbeta complex
GR-hnRNP U complex
YY1-Notch1 complex
GR-hnRNP U complex
POSH-MLK3 complex
NRG1-IKZF4 complex
GR-hnRNP U complex
GR-hnRNP U complex
TIM50a-coilin complex
ADAR1-CDK2 complex
Axin-SMAD3 complex
ADAR1-CDK2 complex
LIN9-BMYB complex
FN1-TGM2 complex
ADAR1-CDK2 complex
SCRIB-APC complex
ADAR1-CDK2 complex
TRAF2-MALT1 complex
ADAR1-CDK2 complex
GR-hnRNP U complex
DLP1-hFIS1 complex
Axin-SMAD3 complex
ASPP1-SAM68 complex
TIM50a-coilin complex
ITGA6-ITGB4 complex
CCND1-CDK4 complex
TIM50a-coilin complex
RFC2-Rialpha complex
TRPC3-TRPC4 channel complex, redox-sensitive
ADAR1-CDK2 complex
AGO2-FXR1-TNF(alpha)ARE-RNP complex
ADAR1-CDK2 complex
ADAR1-CDK2 complex
ADAR1-CDK2 complex
ADAR1-CDK2 complex
ASPP1-SAM68 complex
ASPP1-SAM68 complex
ADAR1-CDK2 complex
ADAR1-CDK2 complex
MAD-MAX complex
ING5 complex
Apoptosome
p33ING1b-HDAC1 complex
TERF1 homodimer complex
SMAD3-E2F4/5-p107-DP1 complex
TIP60 histone acetylase complex
FHL3 homodimer complex
PRMT2 homo-oligomer complex
TERF1 homodimer complex
Calreticulin oligomer complex
NDC80 kinetochore complex
Axin-PP2A A-PP2A C-GSK3-beta-beta-catenin complex
CASK-LIN7C-APBA1 complex
NRD complex (Nucleosome remodeling and deacetylation complex)
NRD complex (Nucleosome remodeling and deacetylation complex)
CAS-SRC-FAK complex
PGAM5-KEAP1-NRF2 complex
SHARP-CtBP complex
TAK1 complex
ING4 complex (ING4, MYST2, C1orf149, PHF15)
Y14-Magoh complex
ACT-CREB complex
IFP35-NMI complex
CNTF-CNTFR-gp130-LIFR complex
AP4 adaptor complex
GINS complex
TFIIIC containing complex
0
0.2
0.4
0.6
0.8
1
1.2

Figure 3.9 Cluster Similarity of Complex between original & filtered graph

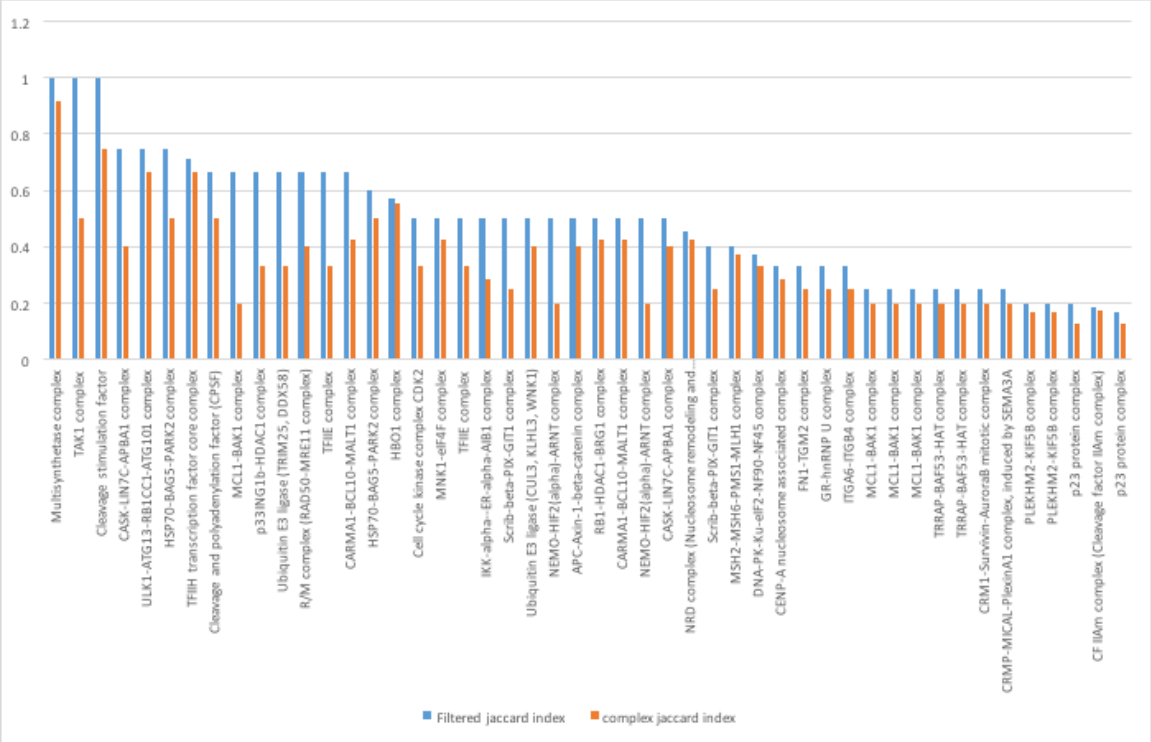
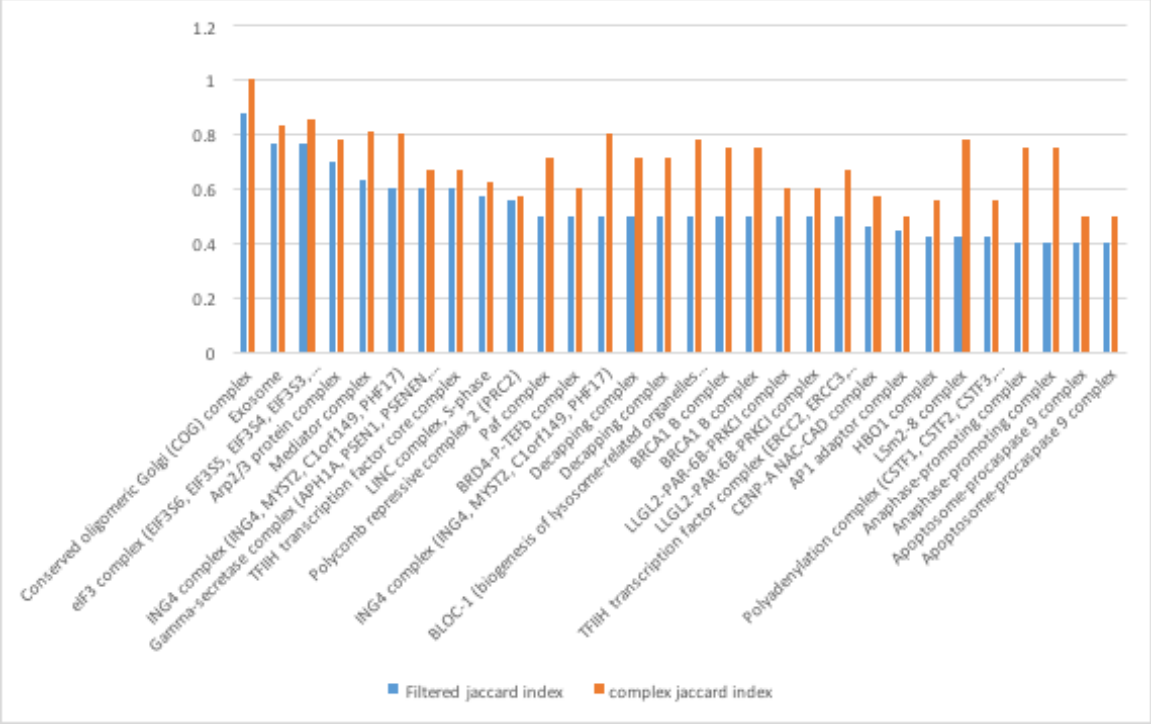


Figure 3.10 Cluster dissimilarity of complex between original & filtered graph



3.2 Chordal Graphs

As discussed in the background section, Chordal graphs are the graphs whose cycles size never be larger than three. If these Chordal graph has a cycle of size more than three, then it should have a chord which is not part of that cycle. This chordal property maintains the triangular structure throughout the graph. In this thesis, we have implemented the algorithm to find maximal chordal subgraph from the given graph.

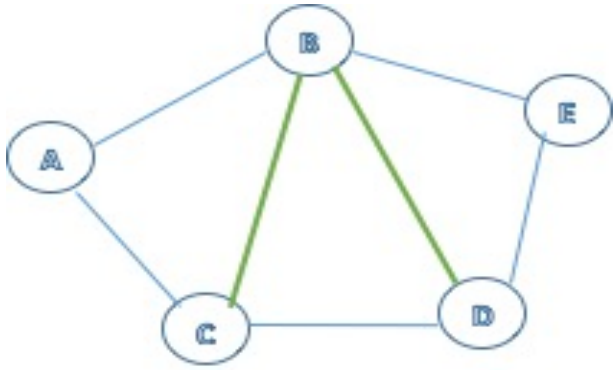


Figure 3.11 Example of simple Chordal graph

In the Figure 4.1, the two green lines B to C and B to D are the chords which present in the cycle (A, B, E, D, C, A) of size more than three and divide it into smaller cliques whose size is 3.

3.2.1 Algorithm to find Maximal Chordal Graph

Many people worked on the chordal graphs and derived algorithms to find the maximal chordal graph. We have used the maximal chordinality search algorithm to find

the maximal chordal graph [5]. The algorithm will give a maximal chordal subgraph G'= (V, E') of any graph G = (V, E).The algorithm steps are presented below:

Algorithm MAXCHORD

For All vertices v in V, set C(v)= null

Define E'=0

Select any v_0 in V, set $S_k=\{v_0\}$

For all $u \in V\text{-}S_k$ with $\{u,v0\} \in E$

If $C(u) \subseteq C(v_0)$

$C(u) = C(u) \cup \{v_0\}$

$E' = E' \cup \{u, v_0\}$

Select $v_0 \in V\text{-}S_k$ such that

$|C(v_0)| >= |C(v)|$

Set $S_k\text{-}1 = S_k \cup \{v_0\}$

$k = k - 1$

If $k > 1$ then go to step 2, else STOP.

3.2.2 Maximal Chordal Subgraphs of Social Networks

We have implemented the above maximal chordinality algorithm to extract the maximal chordal subgraph of a given graph. We have used some social network karate as input to compute the maximal chordal subgraph. The original graph has 34 vertices and 78 edges. The resultant maximal chordal subgraph has 54 edges. We have also

compared various properties between the original graph and the resultant subgraph and specified the results below.

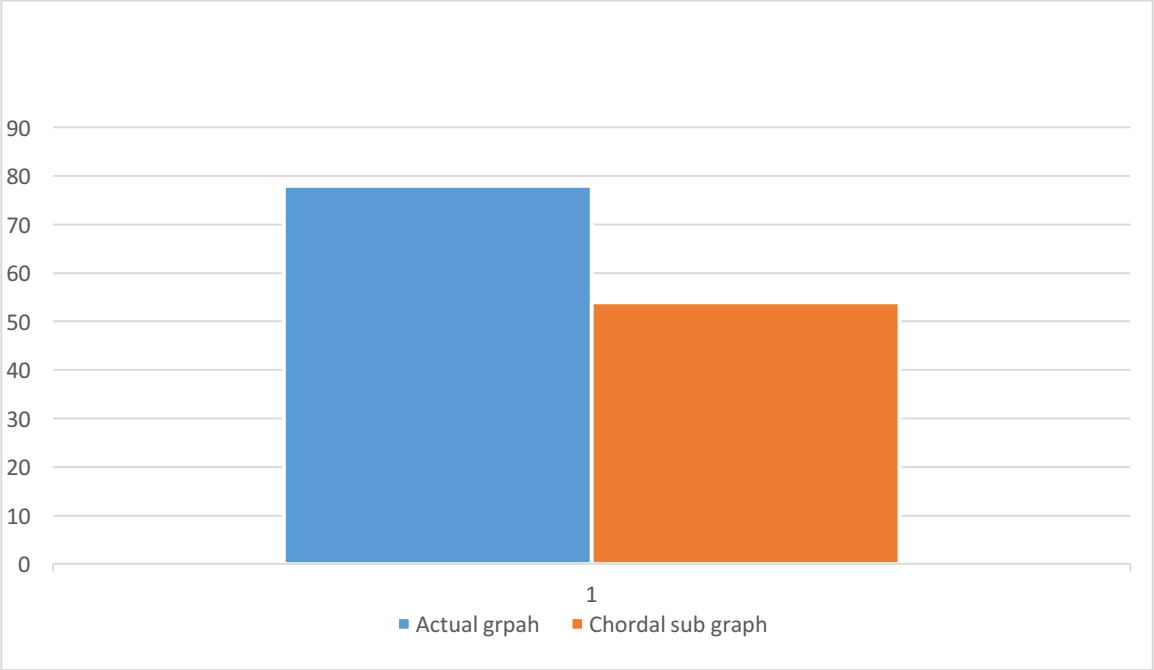


Figure 3.12 Original graph vs Chordal subgraph

3.2.3 Algorithm to find a clique of size n

Finding a clique of size n is a NP-Hard problem. When the given input graph is chordal in nature, then it becomes NP-complete. In this thesis, we have written an algorithm to find the clique of all sizes. Since we have generated chordal subgraphs from the given graph in previous chapter, we have used those filtered chordal social networks as input for our below algorithm to find the cliques of all sizes.

Algorithm to find clique of size n

Input : Chordal graph

Output : Cliques of all sizes

Take a chordal graph as input graph

Find the clustering coefficient of all the vertices

Remove all the vertices and its neighbors whose Clustering Coefficient is 1

The removed vertex and its neighbors will form a Clique

The captured clique size is equal to its neighbor count + 1

Repeat the process for remaining vertices

Continue this process until only one edge is remaining or all the vertices clustering coefficient became zero.

Captured all the cliques of all possible sizes

Using the above algorithm, we have got different sizes of cliques from social networks such as karate, dolphin, lemis, polbooks, adjnoun, football and celeganneural. The experimental results of the algorithms have been presented in the next chapter.

Chapter 4

4.Experimental Results

4.1 Clique size vs No of cliques

In this section, we have demonstrated the availability of cliques of different sizes from different social networks such as karate, dolphin, Lemis, Polbooks, Adjnoun, football Celegan neural, as-in, Netscience, Condmat, Power, Hep and Astro. Below pictures represent the Clique size in the X axis and no of cliques in the Y axis. Most common clique size among all social networks we experimented is 3. Also the largest cycle we found from our experiment is 9.

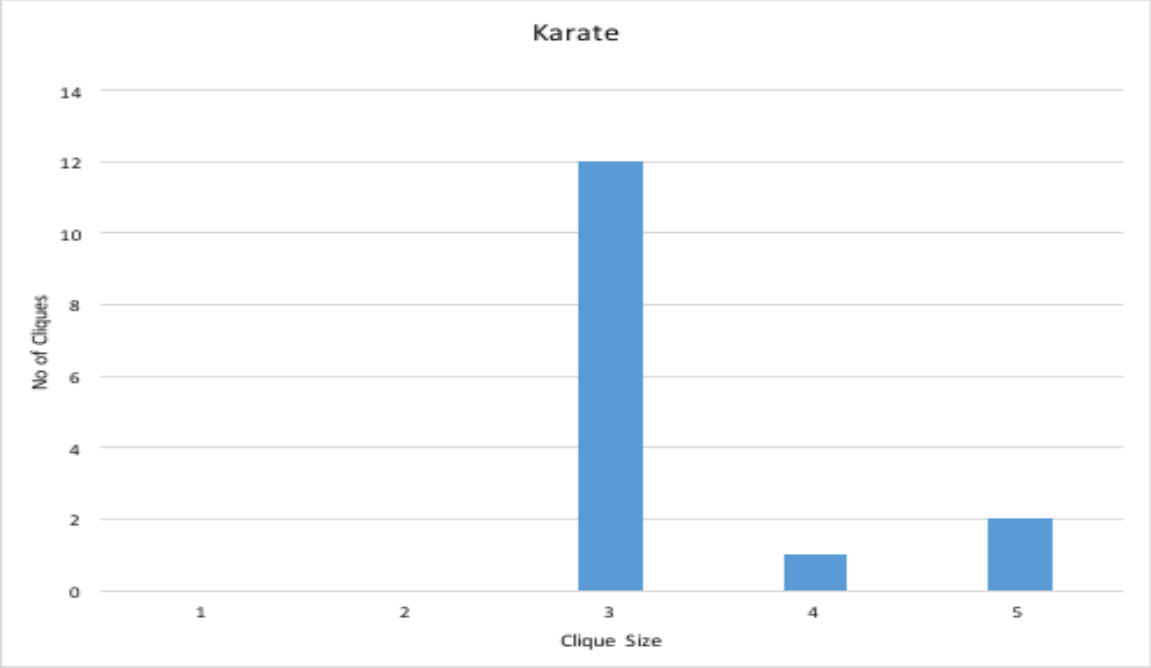
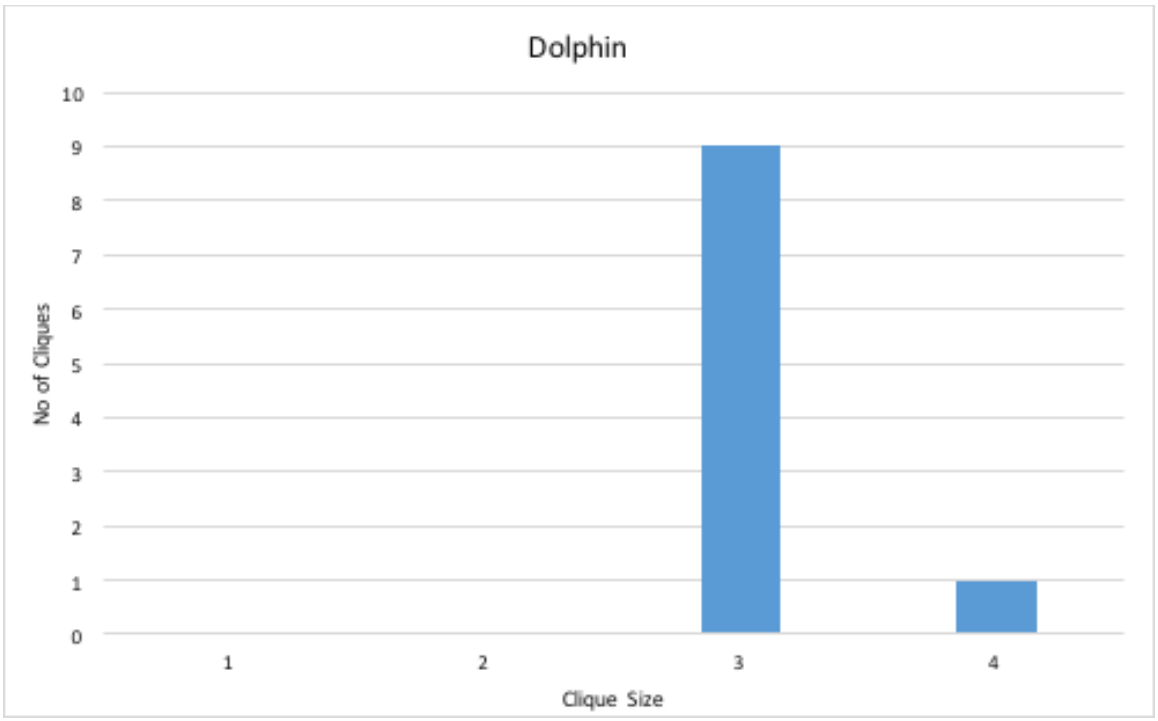
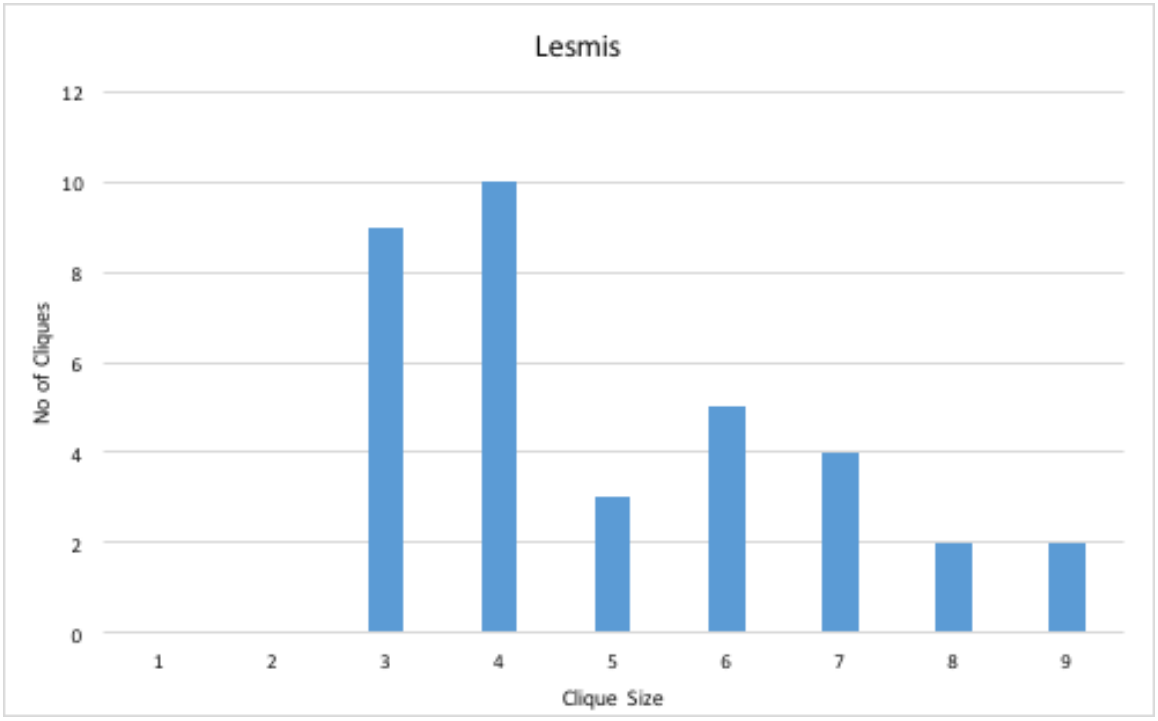
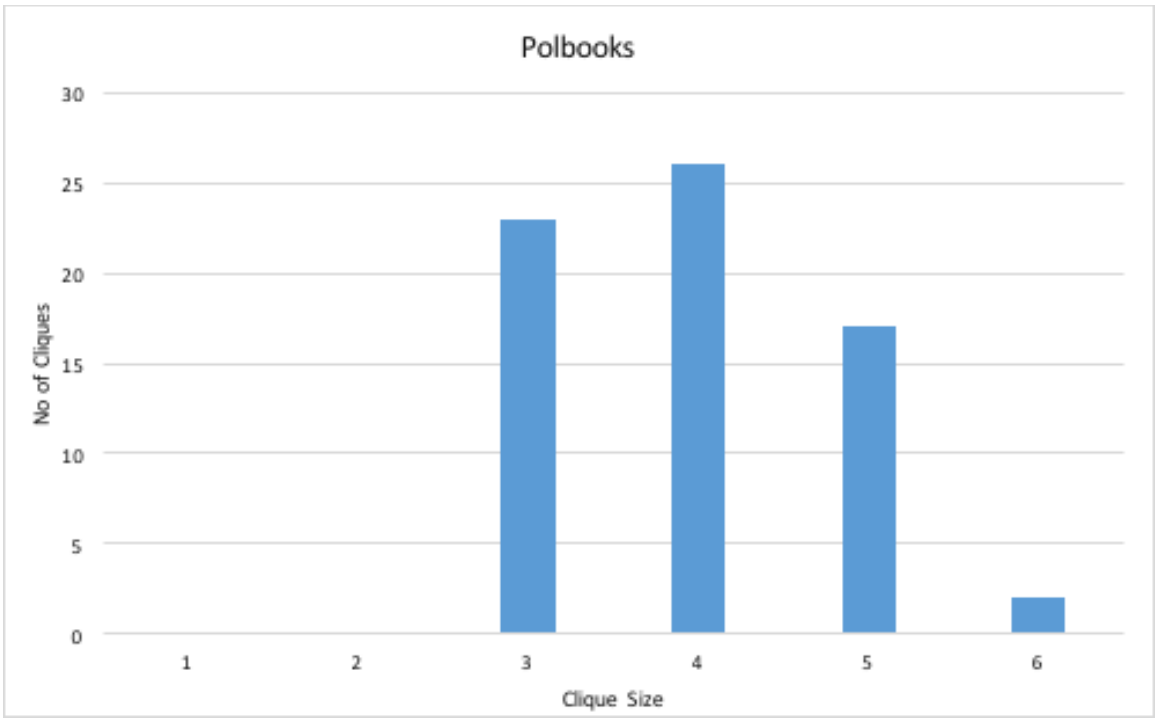
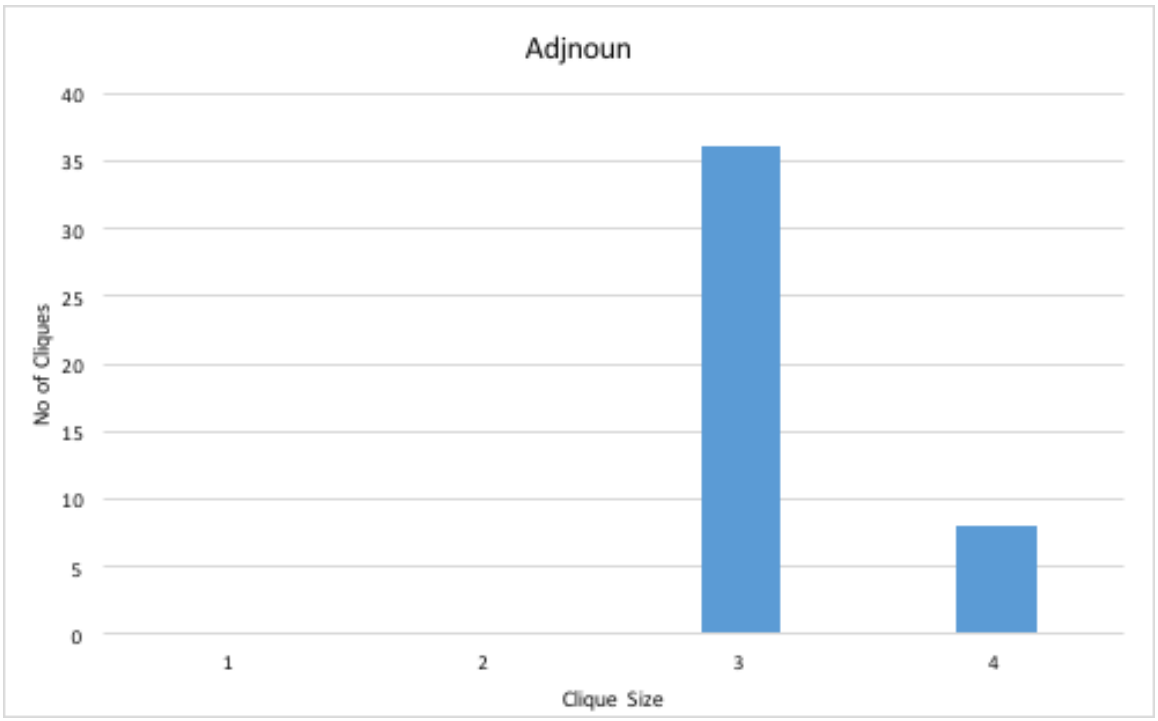


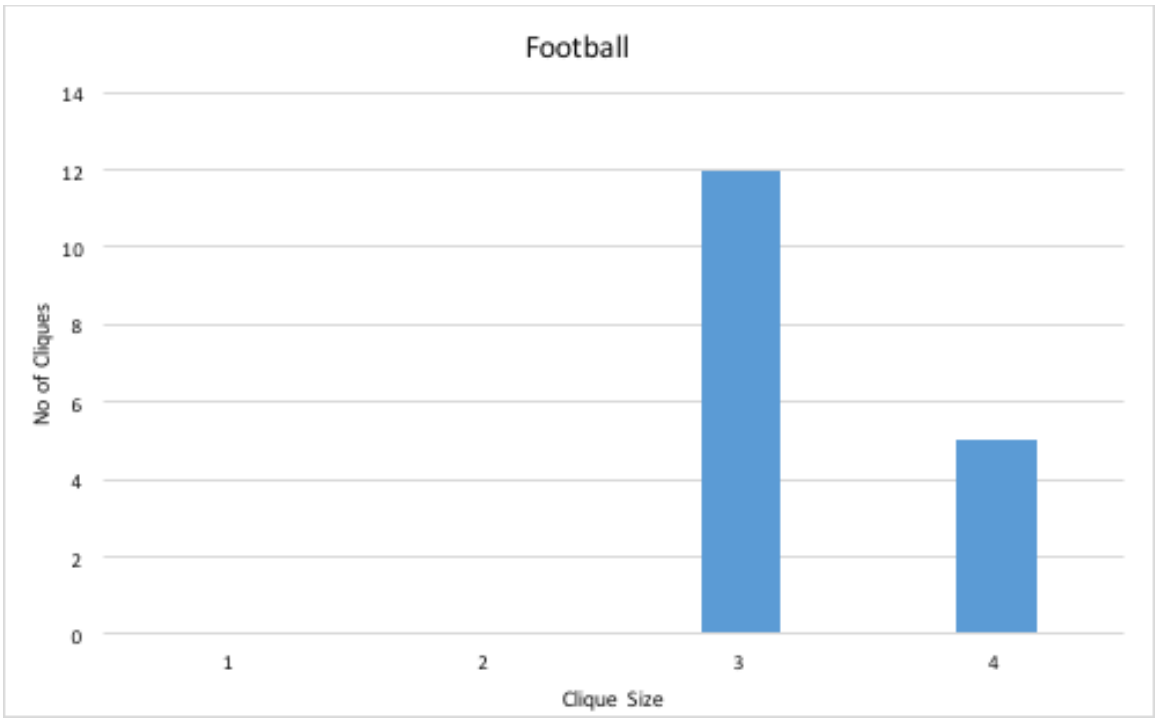
Figure 4.0.1 Clique size vs number of cliques.

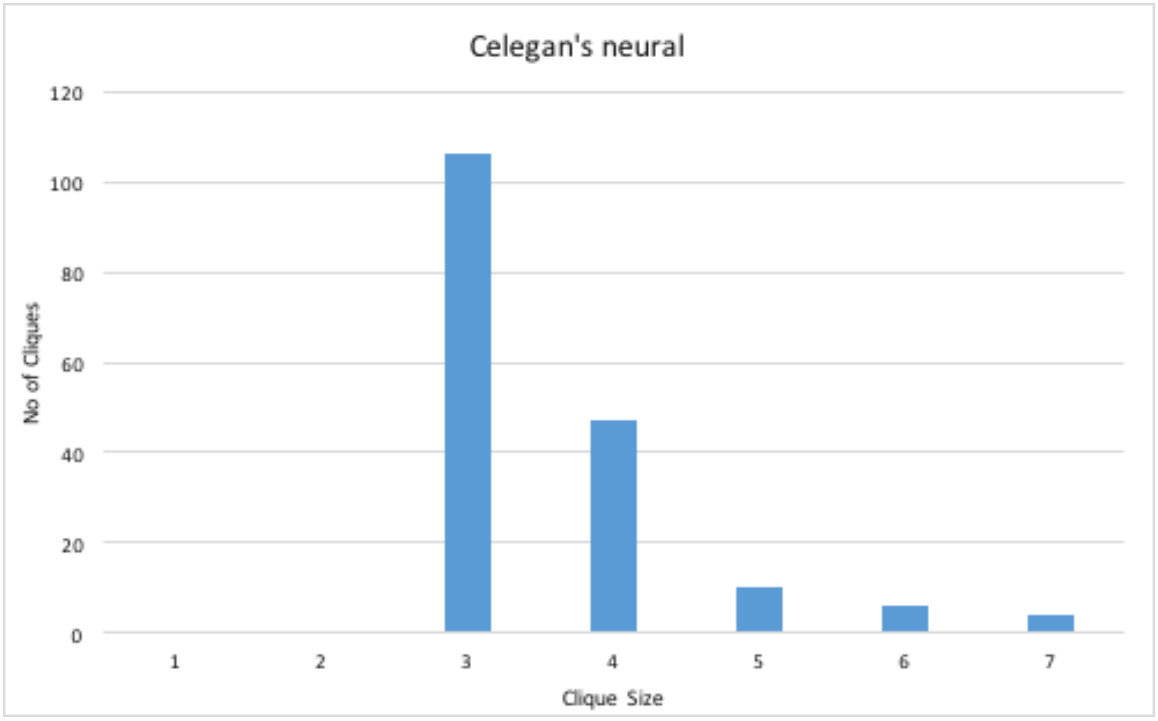

Dolphin
No of Cliques
10
9
8
7
6
5
4
3
2
1
0
1
2
3
4
Clique Size

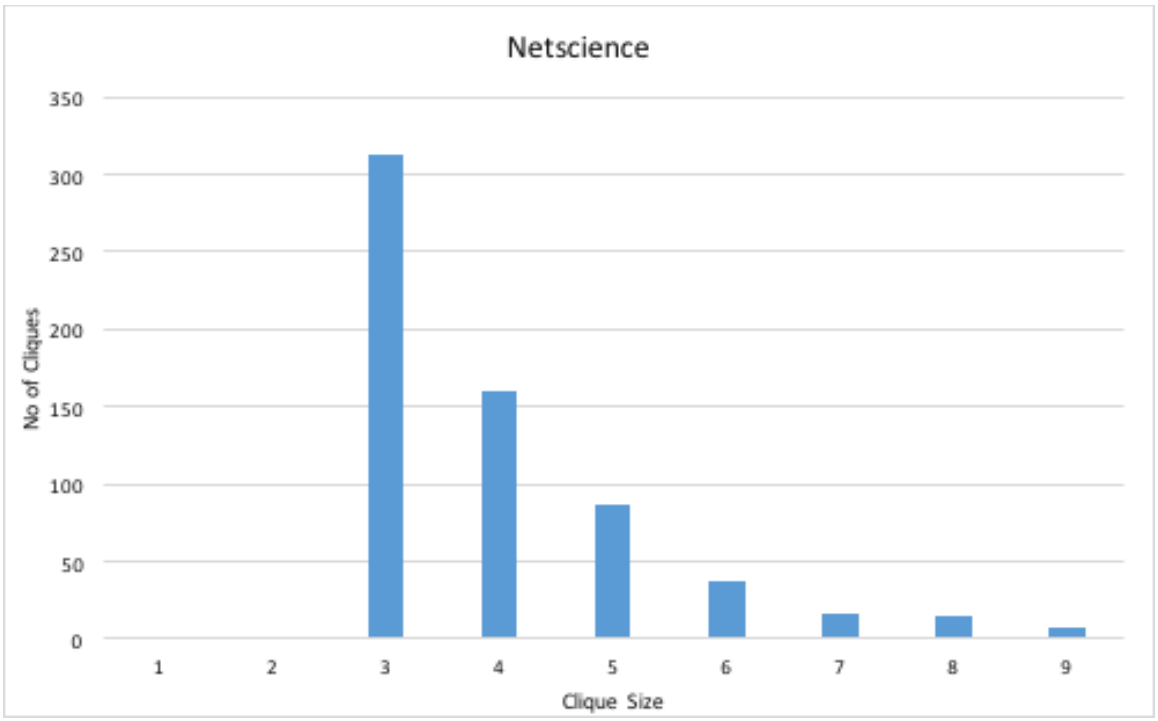

Lesmis
No of Cliques
12
10
8
6
4
2
0
1
2
3
4
5
6
7
8
9
Clique Size

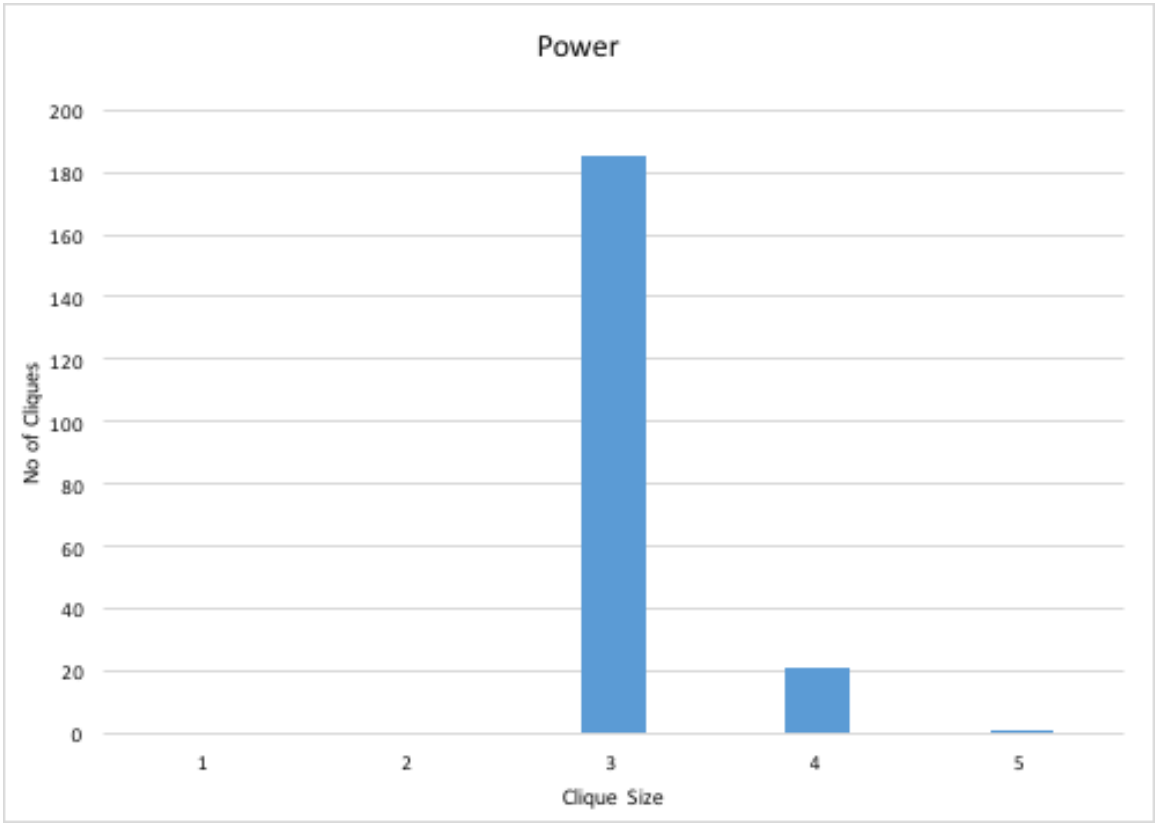

Polbooks
30
25
20
15
10
5
0
No of Cliques
1
2
3
4
5
6
Clique Size

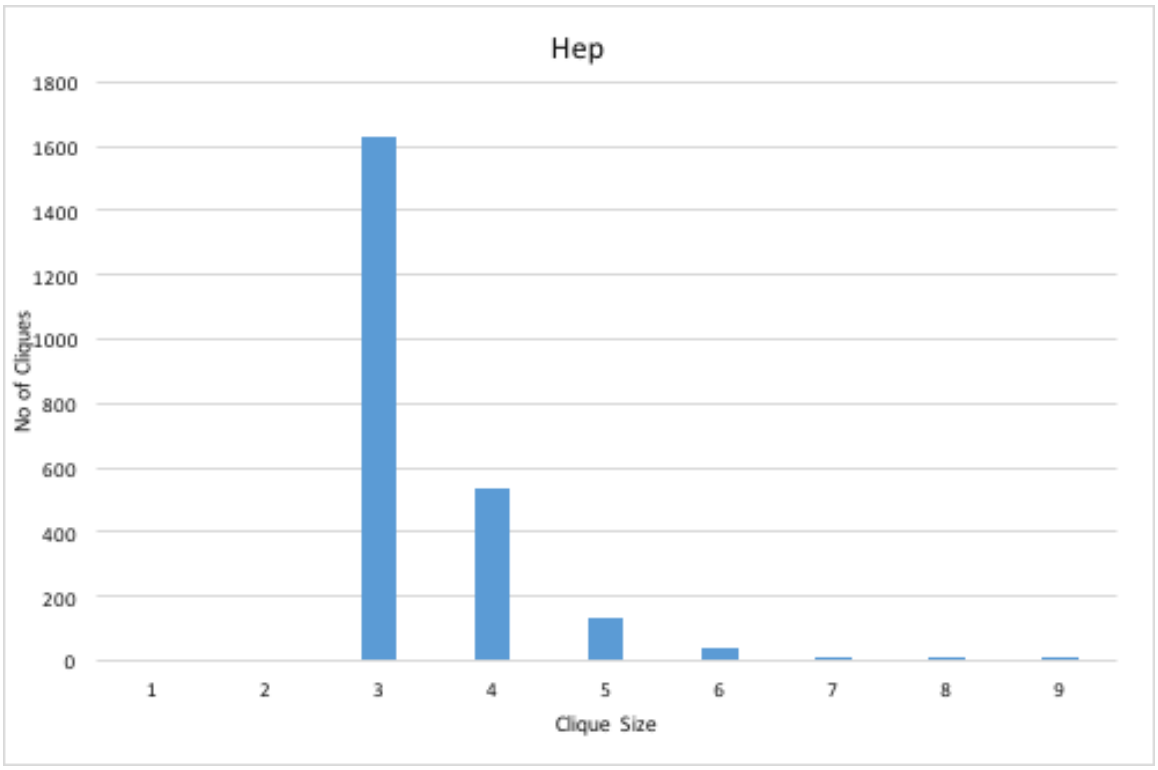

Adjnoun
40
35
30
25
20
15
10
5
0
No of Cliques
1
2
3
4
Clique Size

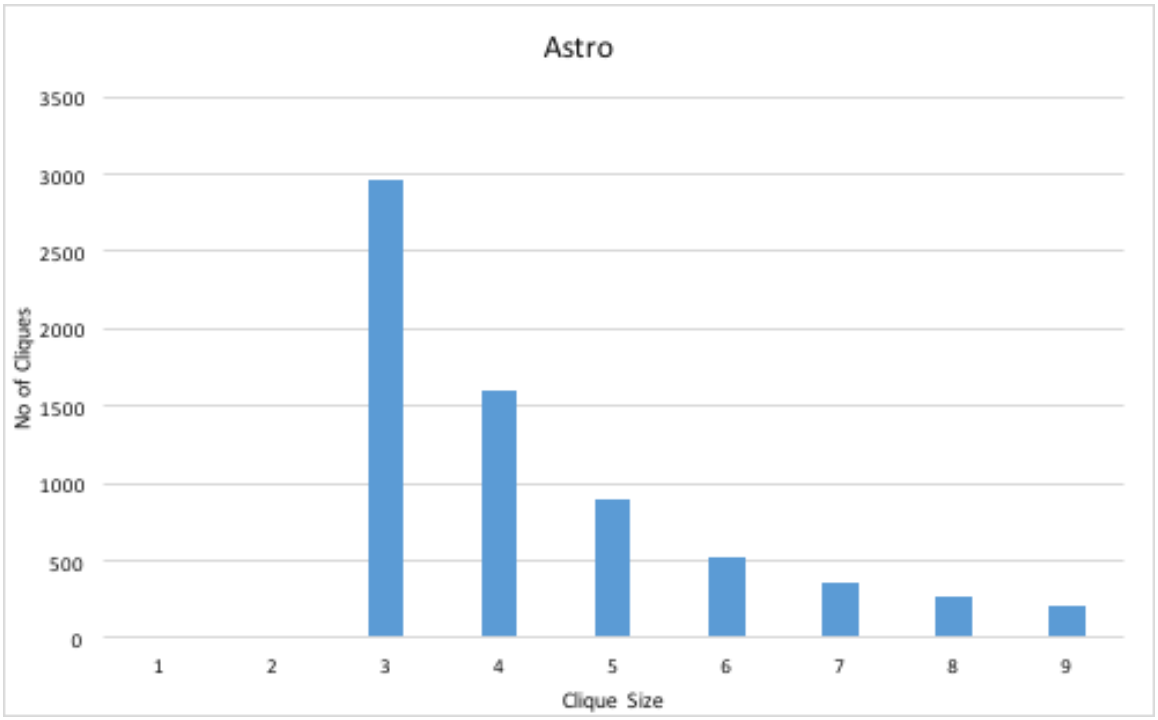

Football
14
12
10
8
6
4
2
0
No of Cliques
1
2
3
4
Clique Size

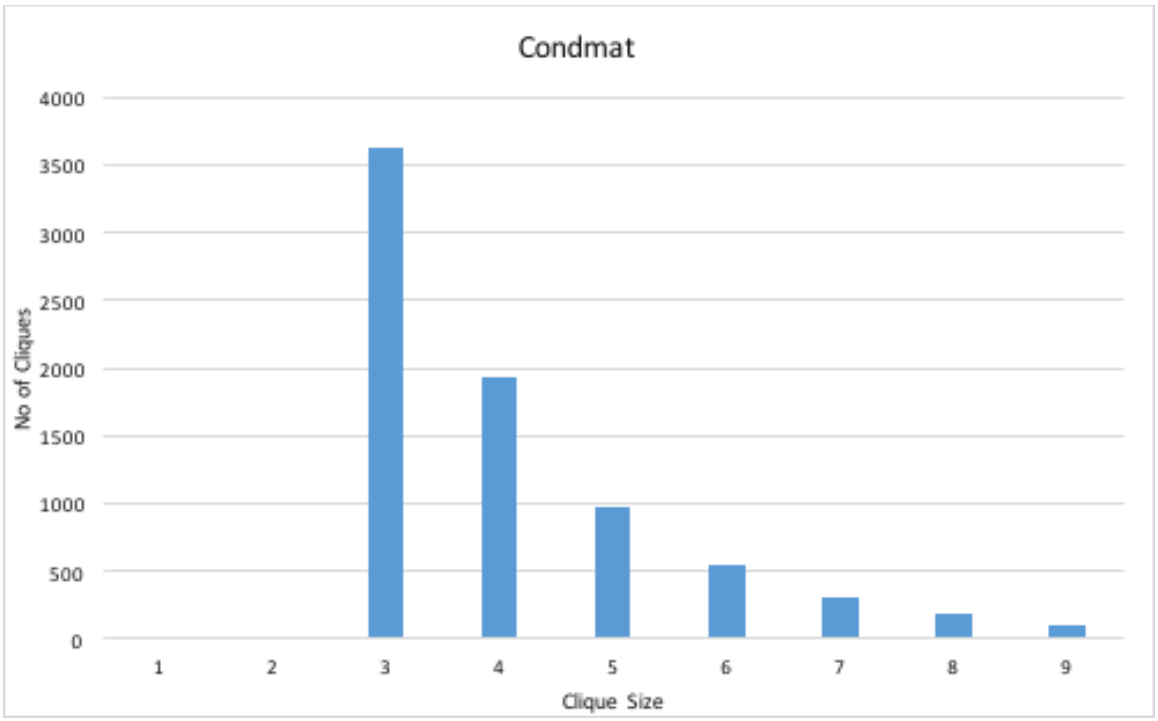

Celegan's neural
120
100
80
60
40
20
0
No of Cliques
1
2
3
4
5
6
7
Clique Size

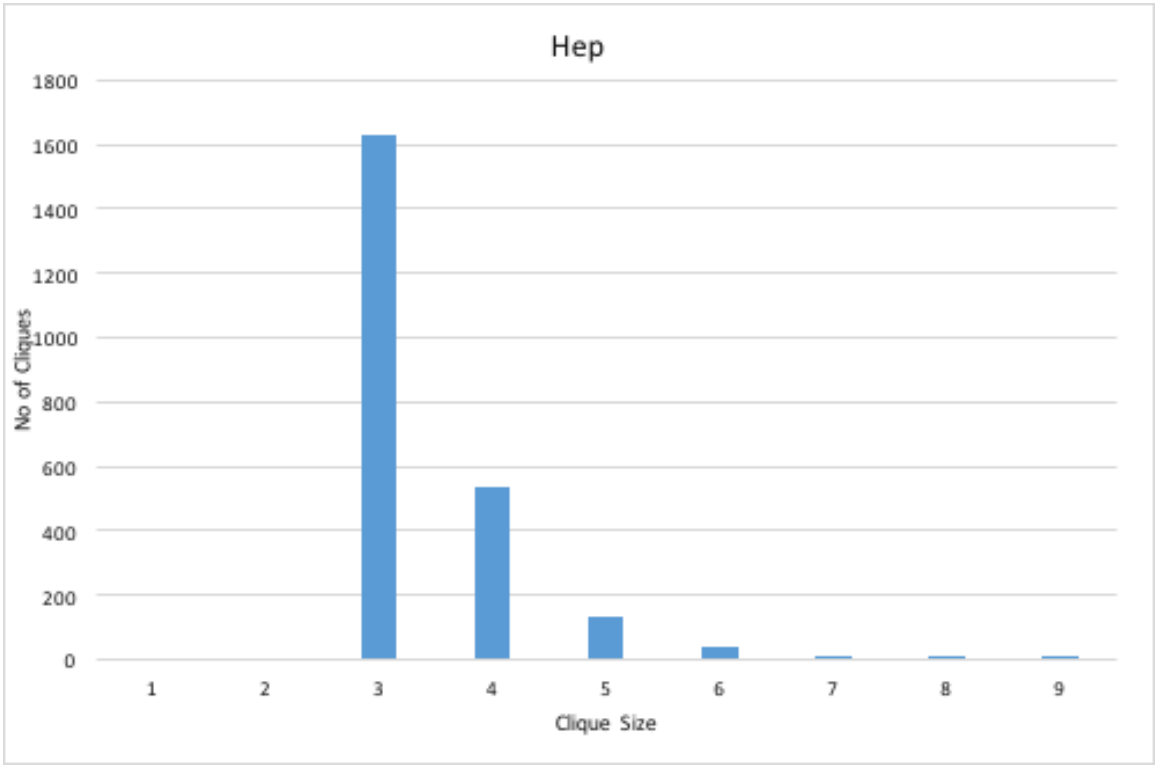

Netscience
350
300
250
200
150
100
50
0
No of Cliques
1
2
3
4
5
6
7
8
9
Clique Size

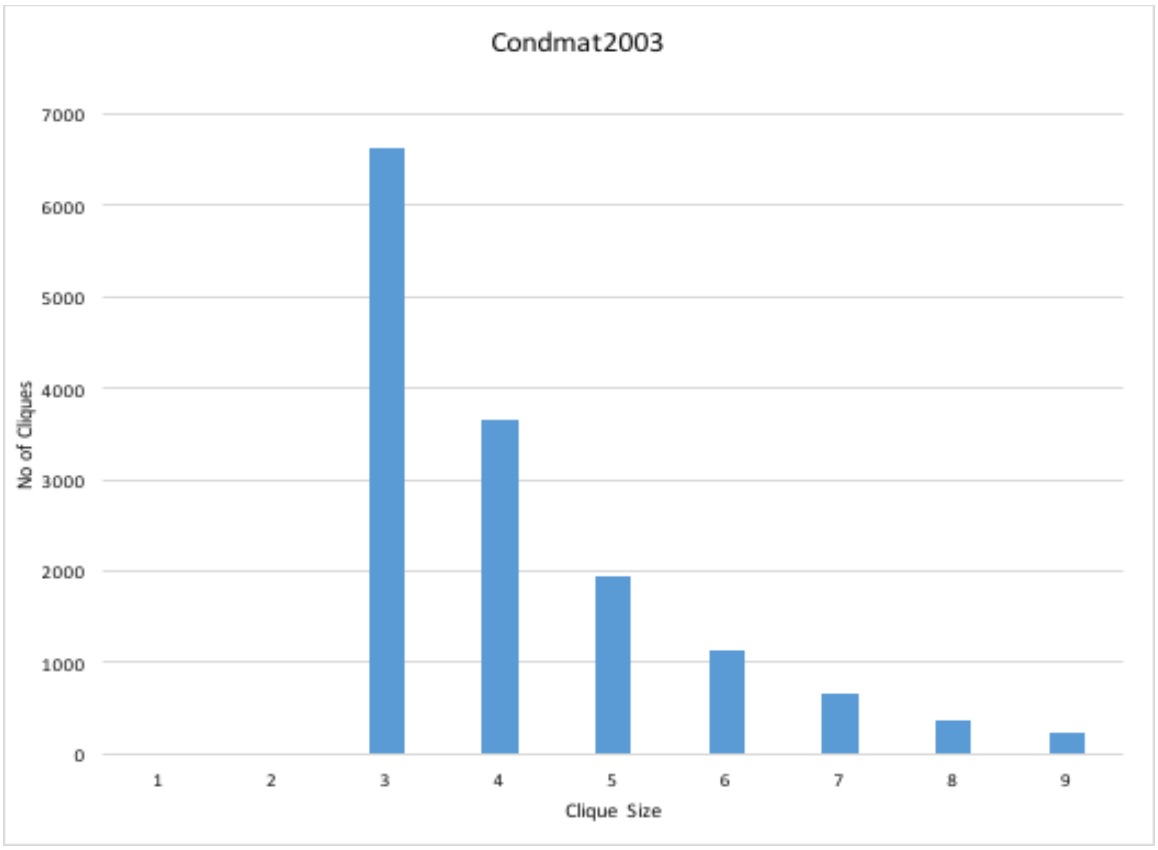

Power
200
180
160
140
120
100
80
60
40
20
0
No of Cliques
1
2
3
4
5
Clique Size

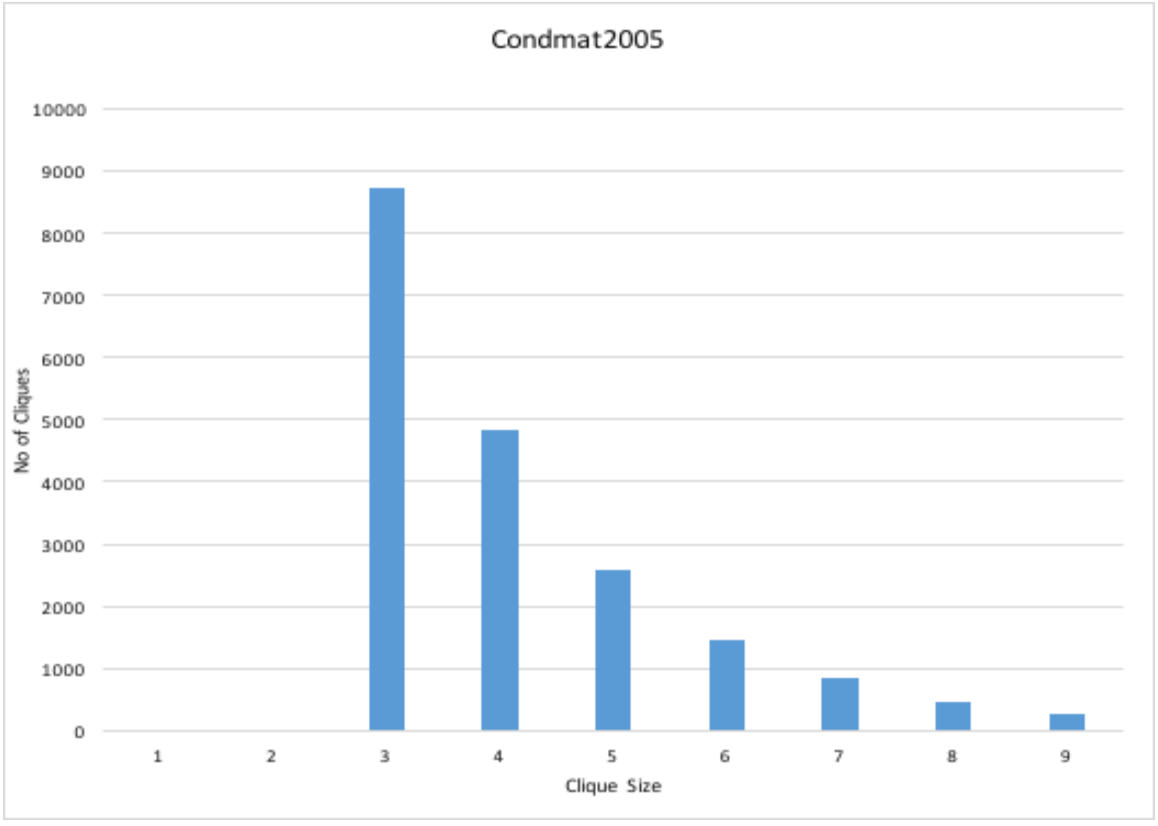

Hep
1800
1600
1400
1200
1000
800
600
400
200
0
No of Cliques
1
2
3
4
5
6
7
8
9
Clique Size


Astro
3500
3000
2500
2000
1500
1000
500
0
No of Cliques
1
2
3
4
5
6
7
8
9
Clique Size


Condmat
4000
3500
3000
2500
2000
1500
1000
500
0
No of Cliques
1
2
3
4
5
6
7
8
9
Clique Size


Hep
1800
1600
1400
1200
1000
800
600
400
200
0
No of Cliques
1
2
3
4
5
6
7
8
9
Clique Size

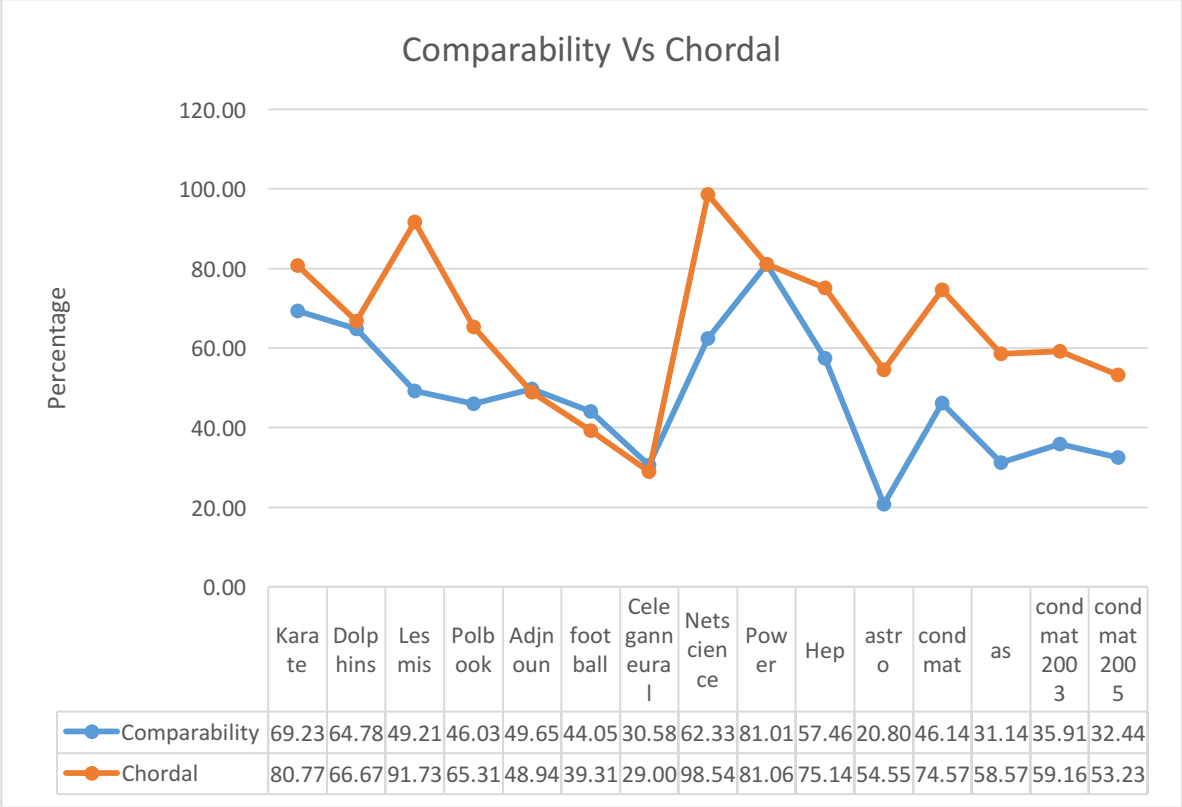

Condmat2003
No of Cliques
7000
6000
5000
4000
3000
2000
1000
0
1
2
3
4
5
6
7
8
9
Clique Size


Condmat2005
No of Cliques
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1
2
3
4
5
6
7
8
9
Clique Size

4.2 Comparison between chordal and comparability

Input Graph		Comparability Graph		Chordal Graph	
Graph	**No of edges in Graph**	**Comparability Graph**	**Percentage**	**Chordal Graph**	**Percentage**
Karate	78	54	69.23	63	80.76
Dolphins	159	103	64.77	106	66.66
Lesmis	254	125	49.21	233	91.73
Polbook	441	203	46.03	288	65.30
Adjnoun	425	211	49.64	208	48.94
football	613	270	44.04	241	39.31
Celeganneural	2345	717	30.57	680	28.99
Netscience	2742	1709	62.32	2702	98.54
Power	6594	5342	81.01	5345	81.05
Hep	15751	9050	57.45	11835	75.13
astro	121251	25220	20.79	66143	54.55
condmat	47594	21959	46.13	35490	74.56
as	48436	15082	31.13	28370	58.57
condmat2003	120029	43099	35.90	71009	59.15
condmat2005	175693	56987	32.43	93516	53.22

Figure 4.0.2 Comparison between chordal and comparability



	Karate	Dolphins	Lesmis	Polbook	Adjnoun	football	Celegans neural	Netscience	Power	Hep	astro	condmat	as	condmat2003	condmat2005
Comparability	69.23	64.78	49.21	46.03	49.65	44.05	30.58	62.33	81.01	57.46	20.80	46.14	31.14	35.91	32.44
Chordal	80.77	66.67	91.73	65.31	48.94	39.31	29.00	98.54	81.06	75.14	54.55	74.57	58.57	59.16	53.23

Figure 4.3 Comparison graph between different subgraphs.

Conclusion & Future work

We developed and implemented an algorithm to find the maximal comparability subgraph from the large scale networks. We have also showed that the subgraphs are relatively close to the chordal subgraph in terms of size. we have also presented the analysis of the comparability filter output with various random vertex ordering. Also our experimental results on protein protein interaction network shows that the comparability filter retains some of the complexes same as original and some of the complexes with high value in comparability subgraph than the original graph. We have developed and implemented the algorithm to find the clique of size n.

As part of future work, we have planned to update our comparability subgraph algorithm as a parallel adaptive algorithm to work in high performance computer so that we can process very large networks in less time. We have also planning to work on finding the largest comparability subgraph (maximum comparability).

References

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

[2] J. C. Miller and J. M. Hyman. Effective vaccination strategies for realistic social networks. Physica A.386,780-785(2007).

[3] D. A. Bader, G. Cong. A Fast, Parallel Spanning Tree Algorithm for Symmetric Multiprocessors.18th International Parallel and Distributed Processing Symposium. IPDPS'04.

[4] G. Cong, G. Almasi, V. Saraswat. A Fast PGAS Implementation of Distributed Graph Algorithms. Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC'10. 2010.

[5] V. Agarwal, F. Petrini, D. Pasetto and D. A. Bader. Scalable Graph Exploration on Multicore Processors. Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. SC'10. 2010

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

[7] P.M. Dearing, D.R. Shier, D.D. Warner Maximal chordal subgraphs, Discrete Applied Mathematics 20 (1988)

[8] J. Leskovec and C. Faloutsos. Sampling from large graphs. Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining,

KDD'06. 2006.

[9] V. Krishnamurthy, M. Faloutsos, J.-H. Chrobak, M. Cui, L. Lao, and A. G. Percus. Sampling large internet topologies for simulation purposes. Computer Networks 51, 2007, 4284–4302.

[10] J. Leskovec, J. Kleinberg, C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, KDD'05. 2005.

[11] Rolf H. Mohring, Algorithmic Aspects of Comparability Graphs and Interval Graphs

[12] K. Duraisamy, K. Dempsey, H. Ali, S. Bhowmick, A noise reducing sampling approach for uncovering critical properties in large scale biological networks

[13] https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/