Certificate

This is to certify that the work presented in this thesis entitled "Techniques for Finding Network Building Blocks in Biological Networks" is the outcome of the investigation carried out by us under the supervision of Professor Dr. Md. Saidur Rahman in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka. It is also declared that neither this thesis nor any part thereof has been submitted or is being currently submitted anywhere else for the award of any degree or diploma.

(Supervisor)

.....

Dr. Md. Saidur Rahman
Professor and Head
Department of Computer
Science and Engineering,
BUET, Dhaka-1000.

(Authors)

.....

Md. Tanzirul Azim
Student No. 0405111

.....

Kishwar Ahmed
Student No. 0405116

Contents

List of Figures

List of Algorithms

Acknowledgments

First of all we would like to thank our thesis supervisor Dr. Md. Saidur Rahman for his intellectual support and important suggestions time to time. This work would not be possible without his guidance and knowledge. As an expert of graph theory he provided us with basic knowledge about many of the graph theoretical problems that eventually lead us to think of a graph theoretical approach to the bio-informatics related problem.

We would like to express our gratitude to Mr. Jawaherul Alam Shuvra, research assistant of this department for his help and suggestion from the very beginning of this work. He helped us to get many resources that were vital for us to carry on the work.

We would also thank all the members of our research group for their valuable suggestions and continual encouragements.

Our special thanks to Mr. Abdullah Al Mueen, PhD student, University of California at Riverside who actually provided us resources initially that lead us to select this topic as a subject of study.

Our parents also helped us to their best efforts. We highly appreciate their support.

And finally thanks to the Almighty for letting us finish this our work in time and without any obstacles.

Abstract

In biological networks a structural unit is the basic building block that contains the properties that make up the total biological features in an entity. These network building blocks can be detected by transforming the total biological network (like Protein -Protein interaction or PPI) to a graph theory problem. For example there exist several methods to find out the building blocks. Those are approached by detecting frequent occurring sub-graphs in a biological network, or by identifying loosely connected groups of nodes in a graph.

Finding network motifs belong to the first approach. That is finding out the frequent sub-graphs. One technique is to enumerate all sub-graphs and count their occurrences over the network. An improved method is to use sub-graph sampling algorithm. It advances by picking random number of edges until k-node sub graph is found out where k is the number of nodes in a sub-graph. Also there are techniques like label propagation. To detect non-overlapped sub-graphs termed as network modules edge-betweenness may be applied. That is to remove those edges that bind the loosely connected sub-graphs. One technique may be to measure all pair shortest path algorithm and remove the most frequently used edges in the network.

We propose a Maximum flow-Minimum cut based algorithm that model the graph as a flow network and finds the sub-grpahs by repeatedly partitioning the graph by finding out the minimum cut and removing those cut-edges .

Chapter 1

Introduction

Network building blocks are densely connected sub-graphs in a network. These network building blocks are commonly termed as network motifs, network modules. There exist several approaches to find out network building blocks in a network. We studied some techniques to find out network building blocks in biological networks. One is finding out frequent sub-graphs in a network. This is a very basic method. Network motifs can be found using sampling algorithm, label propagation algorithm and some other algorithms. To find out network modules within a network, hierarchical clustering technique and edge-betweenness algorithm are frequently used. We also propose an algorithm to find out network modules using a Max flow - Min cut based approach.

1.1 Applications of Network Building Blocks

Finding network modules or network motifs within a complex network has some very important applications in both biological, social networks and also to other network prototypes. We present some of the applications in those networks.

1.1.1 Social Network

After the optimal deconstruction of social networks we can find communities with similar properties. Then further studies of the new sub networks can give inside to the following problems:

• **Social network construction:** The rules by which social networks are formed can be answered from analyzing different patterns of sub network received after deconstruction.

• **Social norms and social networks:** How do social norms emerge and evolve? How are social networks affected by the existence of social norms? The answers to these questions can also be given through analyzing different sub networks.

• **Combining network and field effects:** The result of further study can give inside on how are social cascades affected by the interaction of field effects like mass media with spread through social networks.

1.1.2 Biological Network

Finding network modules or network motifs within a biological network also has some very important applications. Some of them are given in the following.

• **Relationship between components:** Different cellular functions such as information storage, processing and execution is carried out by genome, transcription, and metabolome. Although the functional distinction between these organizational levels is not always clear cut, all cellular functions can be described by networks of various components. One way to visualize the complex relationships between these components is to organize them into a simple complexity pyramid in which various molecular components – RNA, genes, proteins etc organize themselves into recurrent patterns such as metabolic pathways and genetic regulatory motifs. So, the contribution of network motifs comes into play when we want to find out complex relationship between molecular motifs.

• **Organization of sub-networks:** The recent deconstruction of networks has successfully uncovered the skeleton and organization of sub-networks. This offers important insights about the assembly and functionality of components and sub-networks.

1.2 Previous Works

Finding network motifs or network modules within a network has been of great research interest to the researchers. They have found many different ways in the past recent years. In this section we briefly discuss some of them.

1.2.1 Sub-graph Techniques

This technique consists of finding frequently occurring sub-graphs in a network. These sub-graphs are called network motifs. Definition of network motif is given in the following chapter. In short, Network motifs are the structural unit of a complex network. Though they have a shortcoming of not being partitioned. In biological network they have been of great interest. Several methods have been found to detect such network motifs. For example in their papers Shen-Orr *et al.* [Alo02] showed a method to find network motifs in the transcriptional regulation network of Escherichia Coli. Sub-graphs can be found by using a very basic method or using Sub-graph Sampling approach which are discussed in the third chapter.

1.2.2 Hierarchical Clustering and Edge - Betweenness Approach

In this approach we find network modules. The first method is Hierarchical clustering. In this method we build the network building blocks by adding edges to the node pairs according to a certain weight order. But this technique has a shortcoming that if a vertex is connected to the network by a single edge then the algorithm fails.

Another technique is known as Edge-betweenness algorithm. According to this algorithm edge-betweenness of graphs is used to isolate the partitioned sub-graph from the network. In biological networks (i.e protein - protein interaction network) this algorithm can be used to classify similar groups of proteins based on their structural characteristic.

1.3 Our Proposed Algorithm

We propose an algorithm named Max Flow-Min Cut approach in this paper. A similar algorithm was being proposed by Flake *et al.* [Flake00] which is known as Max Flow based approach. Max Flow based approach adopts a page-oriented framework, that is uses a page on the Web as a unit of information, like other methods including HITS and trawling [Flake00]. According to our algorithm we can find all the network modules within a network, which in case of Max Flow based approach by Flake *et al.* is not possible. We can also take input of the size of the network module as input which is also different from that of the algorithm by Flake *et al.* This algorithm can be used extensively in various types of networks such as biological network or community searching. All the features of this algorithm will be discussed extensively in fourth chapter of this thesis paper.

1.4 Conclusion

Both network motif and network module have great importance in various networks like biological and social networks. Some of them have been highlighted in this chapter. Many algorithms regarding finding network motifs and network module have been found in the past. Some of them are discussed in brief. Also the main idea of our proposed algorithm to find out network module within a network has been given in short.

Chapter 2

Preliminaries

This chapter mainly focuses on the terminologies that will be needed throughout the next chapters. Section 2.1 deals with the basic terminologies. In this section we mainly focus on the network related definitions such as biological network, network motifs, network modules etc as they will be used in next chapters. Section 2.2 gives description of some types of network motifs and also their functions.

2.1 Basic Terminology

All the basic definitions of the terms that will be used throughout the paper are given below. Mainly graph related terms are discussed elaborately as they are necessary for our thesis. Complexity of algorithms is also discussed.

2.1.1 Graph

A graph mainly consists of a finite set of ordered pairs. These pairs are called edges or arcs and the entities in the pairs are called nodes or vertices. A graph mainly depicts relationship between two or more variables. An example of a Graph with 7 nodes and 9 edges is given in Figure 2.1(a).

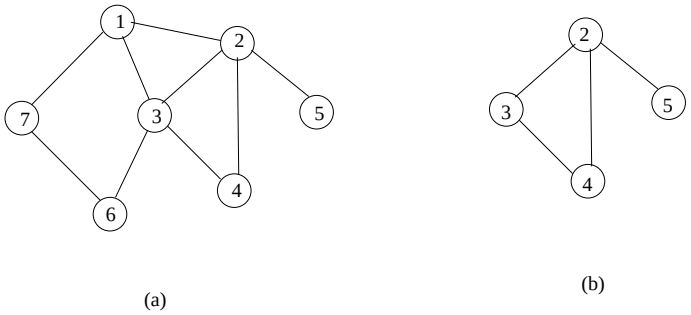

Figure 2.1: (a) A graph with 7 nodes and 9 edges, and (b) a sub-graph of the graph in (a).

2.1.1.1 Sub-graph

A sub-graph of a graph G is a graph whose vertex set is a subset of that of G and whose adjacency relation is a subset of that G restricted to this subset. An example of a Sub-graph is given in Figure 2.1(b).

2.1.1.2 Directed and Undirected Graph

A directed graph G is a pair(V, E) where V is a finite set and E is a binary relation on V. The set V is called the vertex set of G and its elements are called vertices. The set E is called the edge set of G, and its elements are called edges. An example of a directed graph is given in Figure 2.2(a).

In an undirected graph $G = (V, E)$ the edge set E consists of unordered pairs of vertices rather than ordered pairs. That is, an edge is a set u, v where $u, v \in V$ and $u \neq v$. An example of a undirected graph is given in Figure 2.2(b).

2.1.2 Biological Network

A biological network can be defined as a computer system that is designed to mimic the biological entities such as DNA, RNA etc. They were developed to process information simultaneously, adapting and learning from past patterns.

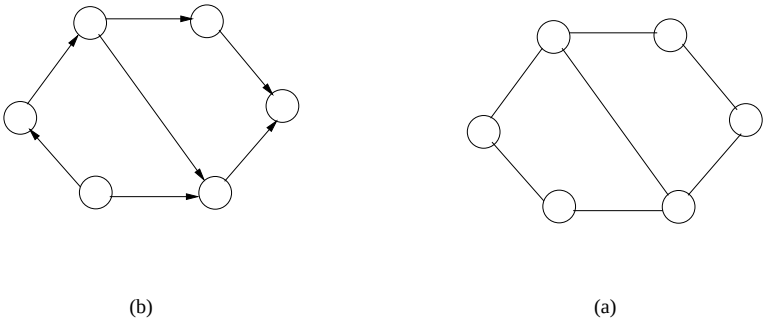

Figure 2.2: (a) A directed graph, and (b) an undirected graph.

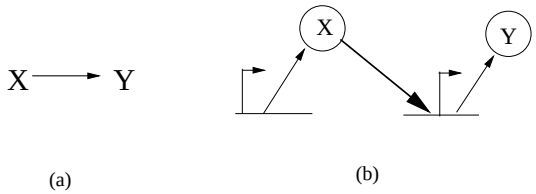

Figure 2.3: Examples of interactions represented by directed edges between nodes. (a) X represents Y, and (b) in a transcription network protein X regulates the production rate of protein Y.

2.1.3 Network Motif

We define network motifs as the patterns of interconnections occurring in complex networks at numbers that are significantly higher than those in randomized networks [Milo02]. So actually Network motifs are patterns (subgraphs) that recur within a network much more often than expected at random. Each type of network (see Figure 2.3) displays its own set of characteristic motifs (ecological networks have different motifs than gene regulation networks, etc). These are taken as small circuits on which the network is built.

2.1.4 Network Module

The sub-graphs of a network which are non-overlapping, unlike network motifs which may be overlapping, are called network modules.

Network modules can be used to find different community structure in social network. Not only that, they are largely used in biological networks too. In a protein complex most of the proteins are involved in the same cellular process or have strong connections with their partners. So biological networks consist of different modules with distinct functions. These modules can be referred as network building modules and can be used at a great extent to find out different modules in biological network.

2.1.5 Edge Betweenness

The betweenness of an edge is defined as the number of the shortest paths running through that edge. The edge betweenness concept is used in Girvan-Newman algorithm for community detection. This algorithm focuses on these edges that are least central, the edges that are most "between" communities. These communities are found out by progressively removing edges from the original graph.

2.1.6 Cluster

A cluster is a group of nodes of the graph. The nodes can be grouped in a cluster according to many criteria.

- Groups of neighbors.
- Groups of nodes fully connected (clique).
- Groups of nodes tightly inter-connected
  - Selection of groups of nodes with high density.
  - Partitioning the graph into near-cliques which are sparsely inter-connected.
  - Selection of the nodes presenting more interactions with the cluster nodes than with the other nodes of the graph.

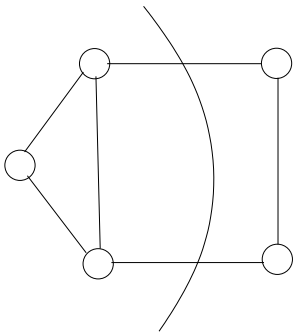

Figure 2.4: An example of minimum cut with cut size 2.

2.1.7 Cut

A cut is a split of the nodes into two disjoint sets S and T, such that s is in S and t is in T. The capacity of a cut (S, T) is, the sum of the capacity of all the edges crossing the cut, from the region S to the region T. It represents the maximum amount of flow that can pass through an edge. Mathematically, they can be defined as given below,

An s-t cut, $C = (S, T)$ is a partition of V such that $s \in S$ and $t \in T$.

The capacity of an s-t cut is defined by $c(S, T) = \Sigma_{v \epsilon V} f_{\mathrm{sv}}$.

2.1.7.1 Minimum Cut

A cut is minimum if the size of the cut is no larger than the size of any other cut. The minimum cut problem is to minimize $c(S, T)$, i.e. to minimize the amount of capacity of an s-t cut. An example of minimum cut with cut size 2 is given in Figure 2.4.

2.1.8 Flow Network

A flow network is a directed graph where each edge has a capacity and each edge receives a flow. A flow must satisfy the restriction that the amount of flow into a node equals the amount of flow out of it, except when it is a source, which has more outgoing flow, or sink which has more incoming flow. Max flow between two nodes is the maximum amount of flow in a graph. An

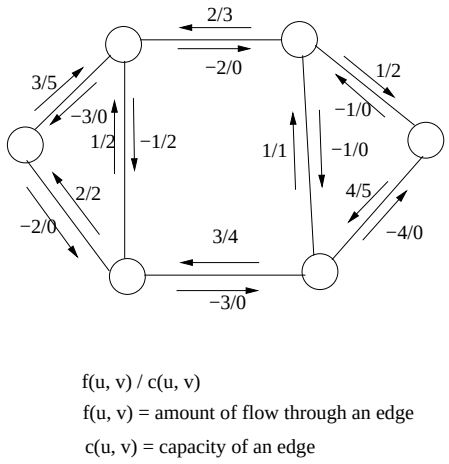

Figure 2.5: Flow network.

example of a flow network is given in the Figure 2.5.

Mathematically, this can be treated as following:

Let $G(V, E)$ be a directed graph, with capacity greater than zero defined for each edge. Let the capacity be $C(u, v)$ for each edge$(u, v) \epsilon E$. If $(u, v) \neq 0$ we have $C(u, v) = 0$. There are two distinguished vertices source s and sink t. A flow network is a real function $f : V \times V \rightarrow R$. Here all nodes u, v has the three properties:

Capacity Constraints: $f(u, v) < c(u, v)$That is the flow along an edge cant exceed its capacity.

Skew symmetry: $f(u, v) = -f(v, u)$.

Flow Conservation: $\sum_{w \epsilon V} f(u, w) = 0$, unless $u = s$or $u = t$.

2.1.9 Complexity of Algorithm

Complexity of an algorithm can be defined as the worst case running time of an algorithm. A run time of an algorithm can take seconds, hours or even years to finish executing. Usually the complexity is a function of input length. Some convenient ways of describing the growth rate of a function and hence the time complexity of an algorithm are given below.

2.1.9.1 Big Oh Notation

Let n be the size of the input and $f(n)$, $g(n)$ be positive functions of n. Then, $f(n)$ is $O(g(n))$ if and only if there exists a real, positive constant C and a positive integer n_0 such that

$f(n) \leq Cg(n) \ \forall \ n \geq n_0$

- Big Oh notation provides an asymptotic upper bound.
- $O(1)$refers to constant time, $O(n)$ indicates linear time; $O(n^k)$ (k fixed) refers to polynomial time; $O(logn)$ is called logarithmic time; $O(2^n)$ refers to exponential time, etc.

2.1.9.2 Big Omega Notation

$f(n)$ is $\Omega(g(n))$ if and only if there exists a real, positive constant C and a positive integer n_0 such that

$f(n) \geq Cg(n) \ \forall \ n \geq n_0$

The Big Omega notation specifies asymptotically lower bounds.

2.1.9.3 Big Theta Notation

$f(n)$is $\theta(g(n))$ if and only if there exist real, positive constants C_1, C_2 and a positive integer n_0 such that

$C_1g(n) \leq f(n) \leq C_2g(n) \ \forall \ n \geq n_0$

The Big Omega notation specifies asymptotically tight bounds.

2.2 Different Types of Network Motifs

Below are some of the common Network motifs and their associated functions[1].

[1] http://en.wikipedia.org/wiki/Network_motif

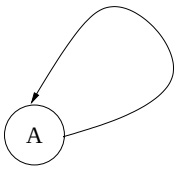

Figure 2.6: Schematic representation of an auto-regulation motif.

2.2.1 Negative Auto-Regulation (NAR)

Negative Auto-Regulation is one of the simplest and most abundant in E. Coli. This motif performs two important functions. They are given below:

- The first function is response acceleration. NAR speeds up the response to signals both theoretically and experimentally.
- The second function is increased stability of auto-regulated gene product concentration against stochastic noise.

2.2.2 Positive Auto-Regulation (PAR)

Positive auto-regulation occurs when a transcription factor enhances its own rate of production. This motif slows down the response time which is opposite to NAR.

2.2.3 Feed-Forward Loop(FFL)

Feed-Forward Loop is very common in gene systems and organisms. The motif consists of three genes and three regulatory interconnections. Since each of the regulatory interactions may be either positive or negative there are eight types of FFL motifs. Two of these types: Coherent type 1 FFL (C1-FFL) and Incoherent type 1 FFL (I1-FFL) are very much frequent in the transcription network of E. Coli and Yeast then the other six types.

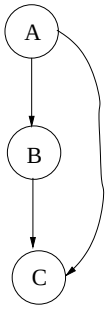

Figure 2.7: Schematic representation of a Feed-forward motif.

2.2.4 Coherent Type 1 FFL (C1-FFL)

In case of C1-FFL all interactions are positive. The motif can provide pulse filtration in which short pulses of signal will not generate a response but persistent signals will generate a response after a short delay.

2.2.5 Incoherent Type 1 FFL (I1-FFL)

The I1-FFL works as both pulse generator and response accelerator. The functions being performed by this type network motif is given below:

- It works as a pulse generator. The two signal pathways if it acts in opposite directions. When the repression is complete this leads to a pulse like dynamics.
- It can also serve as response accelerator in a way which is similar to the NAR motif.
- I1-FFL can generate non-monotonic input function in both a synthetic and native systems.

2.2.6 Multi-Output FFL

This motif can be used to determine the temporal order of gene activation. This was demonstrated experimentally in the flagella system of E.Coli.

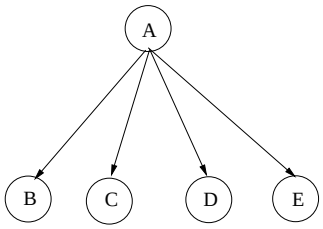

Figure 2.8: Schematic representation of Single-input modules.

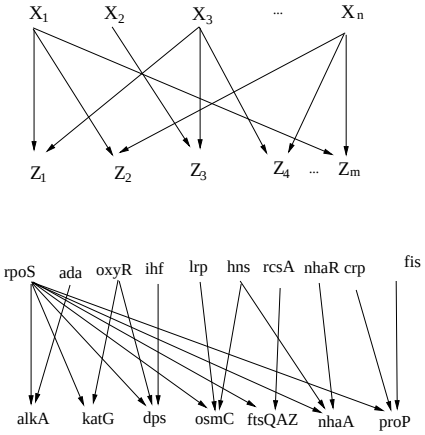

Figure 2.9: Schematic representation of Dense overlapping regions.

2.2.7 Single-Input Module (SIM)

Single-Input Module occurs when a single regulator regulates a set of genes with no additional regulation. By adjusting the strength of the interactions it can create temporal expression program of the genes it regulates. An example of Single-input modules is given in Figure 2.8.

2.2.8 Dense Overlapping Region (DOR)

This motif was found in E.Coli in various systems such as carbon utilization, anaerobic growth, stress response and others. In order to better understand the function of this motif one has to obtain more information about the way the multiple inputs are integrated by the genes. An example of Dense overlapping regions is given in Figure 2.9.

2.3 Conclusion

This chapter has highlighted the very basic definitions that are needed in this thesis. The concept of network motif and network module should help the readers to understand the previous algorithms being used and also the new approach we give in this paper. Different types of network motifs and their functions have also been given to clarify the concept of network motif.

Chapter 3

Network Motifs and Network Modules

3.1 Introduction

The idea of "Network Motifs" was presented to define the structural design principles of complex networks. The networks can be from biochemistry, neurobiology, ecology, and engineering. The networks may be "ecological food webs" or the "genetic networks of *E.coli*. Networks can be also information processing networks, or synoptic connections between the neurons. So network motifs are the universal building blocks of Networks.

3.2 Network Motifs

There has been works to introduce Network Motif concept in biological networks like gene regulatory networks. These networks are built on the genes which are responsive to biological signals in the cell. The network is defined such that genes are nodes, and directed edges represent the control of one gene by a transcription factor (regulatory protein that binds DNA) encoded by another gene. Thus, network motifs are patterns of genes regulating each others transcription rate. When analyzing transcription networks,

it is seen that the same network motifs appear again and again in diverse organisms from bacteria to human. The transcription network of E. Coli and yeast, for example, is made of three main motif families, that make up almost the entire network. The leading hypothesis is that the network motif were independently selected by evolutionary processes in a converging manner [Babu04, Conant03], since the creation or elimination of regulatory interactions is fast on evolutionary time scale, relative to the rate at which genes change [Babu04, Conant03, Dekel05], Furthermore, experiments on the dynamics generated by network motifs in living cells indicate that they have characteristic dynamical functions. This suggests that the network motif serve as building blocks in gene regulatory networks that are beneficial to the organism.

3.3 Previous Works

In the past significant researches have been conducted to identify this structural elements. From the previous researches the idea of *'Network Motif'* [Alo02] came into being. They can be defined as the sub-graphs in the network that occur significantly more often than the number of times they occur in the corresponding random networks [Alo02]]. These motifs are not found only in biological networks but also in other networks such as web communities as social networks. The network motifs in the transcriptional regulation network of E. coli were studied by Shen-Orr *et al.* [Alo02]. According to the authors feed-forward loop, the single input module and the dense overlapping regions were found to be main building blocks. The concept of Composite Network Motifs were proposed by Yeger-Lotem *et al.* [Sattath04]. Transcription-regulation and protein-protein interaction networks show those patterns and they appear significantly more than other random networks. Motifs were identified by them as two, three and four protein motifs. Network Motif Concept has also been used to classify graphs. Milo *et al.* [Alo02] introduced the concept of significance profile which is computed over the

small sub-graphs of the network and is used to cluster different networks. Qiaofeng Yang *et al.* [Qia07] showed that the previous techniques were actually a mechanism of following ideas, (1) they are designed to operate on directed graphs and (2) they are based on the exhaustive enumeration of all the sub-graphs. Hence those methods were named Sub-graph Counting Network Motif (SCNM) approaches.

3.4 Network Motifs Finding Algorithms

Existing algorithms on network motifs rely on exhaustively enumerating all sub-graphs with a given number of nodes in the network. The run-time of such algorithms increases with the network size. A concept of such algorithm was given in U. Alon *et al.* [Alo02].

3.4.1 Traditional Sub-graph Finding Algorithm

The interaction between nodes are presented by Direct edges. As in Figure 2.3. All possible n-node sub-graphs (in the present study, n equals 3 and 4), and the number of occurrences of each sub-graph was recorded by scanning through the network. Numerous type of node sub-graphs are contained in each network (Figure 3.1). The real network is compared with some randomized networks (12–16) and selected patterns are occurred more often in real networks then the randomized ones(Figure 3.2). For a stringent comparison, randomized networks with same single-node characteristics as does the real network are used: Each node in the randomized networks has the same number of incoming and outgoing edges as the corresponding node has in the real network. The comparison to this randomized ensemble accounts for patterns that appear only because of the single-node characteristics of the network (e.g., the presence of nodes with a large number of edges). Furthermore, the randomized networks used to calculate the significance of n-node sub-graphs were generated to preserve the same number of appearances of all (n – 1)-node sub-graphs as in the real network [Kashtan04, Alo02]. This

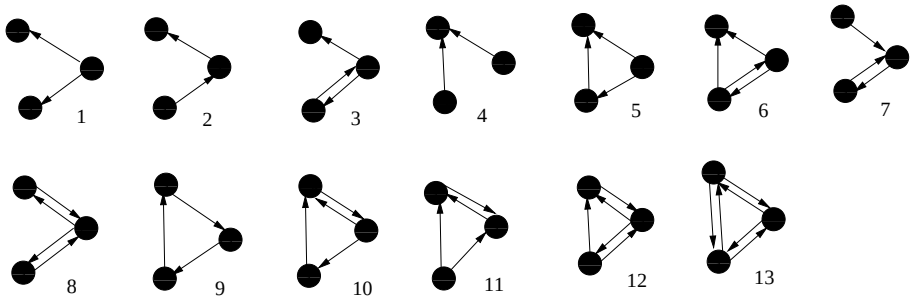

Figure 3.1: All 13 types of three-node connected sub-graphs.

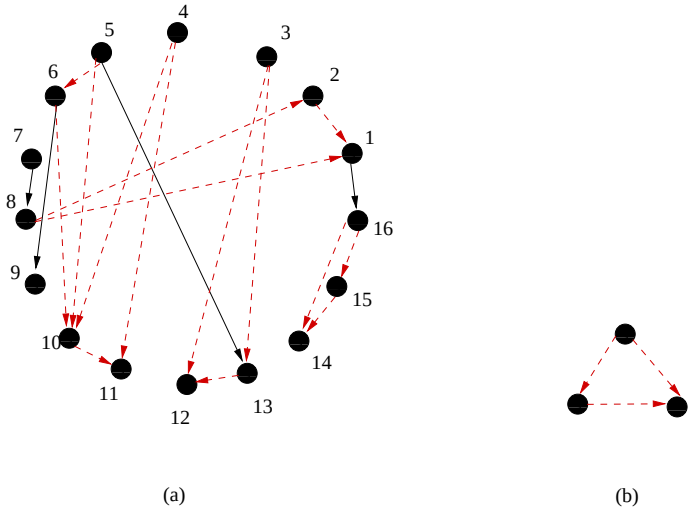

Figure 3.2: Schematic view of network motif detection. (a) A real network, and (b) a network motif with 3 nodes which occurs 5 times in the real network given in (a).

ensures that a high significance was not assigned to a pattern only because it has a highly significant sub-pattern.

3.4.2 Sampling Algorithm

An efficient sampling algorithm was proposed by U. Alon *et al.* [Alo02]. This algorithm is asymptotically independent of the network size. Few samples are needed to detect network motifs reliably.

3.4.2.1 Sub-graph Concentrations

Direct networks with one color of edges and nodes are selected. Let the number of appearances of sub-graph i is N_i. The concentration of n node sub-graphs of type i is shown in equation 3.1.

$$C_i = \frac{N_i}{\sum N_i} \tag{3.1}$$

$\sum_i N_i$ = Total number of n-node connected sub-graphs in the network.

3.4.2.2 Sub-graph Sampling

The samples n-node sub-graphs by picking random connected edges until a set of n-nodes reaches. The method is described below:

- A random edge is picked from the network.
- The sub-graph is expanded iteratively by picking random neighboring edges until the sub-graph reaches n nodes.
- For each random choice of an edge, in order to pick an edge that will expand the sub-graph size by one, prepare a list of all such candidate edges and then randomly choose an edge from that list.
- The sampled sub-graph is defined by the set of n nodes and all the edges that connect between these nodes in the original network.

The procedure is listed in Algorithm 1.

3.4.3 Label Propagation Algorithm

Raghban *et al.* [Raghban07] studied with a new label propagation algorithm that was claimed to be nearly linear. It is a localized algorithm. The basic ideas are:

- Each node is initialized with a unique label.

Algorithm 1 Algorithm for sub-graph sampling.

Definitions: E_s is the set of picked edges.
V_s is the set of all nodes that are touched by the in E_s

Init V_s and $E_s$to be empty sets.
1. Pick a random edge e_1= (v_i, v_j) . Update $E_s = \{e_s\}$, V_s=(v_i, v_j)
2. Make a list $L$of all neighboring edges of E_s.
   Omit from L all edges between members of V_s. If L is empty return to 1.
3. Pick a random edge e =(v_k, v_l) from L.
   Update $E_s = E_s \cup e$, $V_s = V_s \cup (v_k, v_l)$
4. Repeat steps 2-3 until completing n-node sub-graph S.
5. Calculate the probability P to sample S.

- At every iteration of the algorithm, each node adopts a label that a maximum number of its neighbors have, with ties broken uniformly randomly.
- As the labels propagate through the network in this manner, densely connected groups of nodes form a consensus on their labels.
- At the end of the algorithm, nodes having the same labels are grouped together as communities.

3.4.4 Other Algorithms

Szpankowski *et al.* [Koyuturk04] also showed an efficient algorithm that is based on mining pathway substructures. Their work was basically involved in mining metabolic pathways.

3.5 Network Modules

Recent trends are going on to identify building modules of network which are commonly referred as "network modules". To identify network modules within a network we mainly discuss an algorithm which is based on the notion of Edge Betweenness [GN02]. This algorithm finds highly connected network modules that occur more frequently than those found in the corresponding random networks. This method is highly used for detecting community

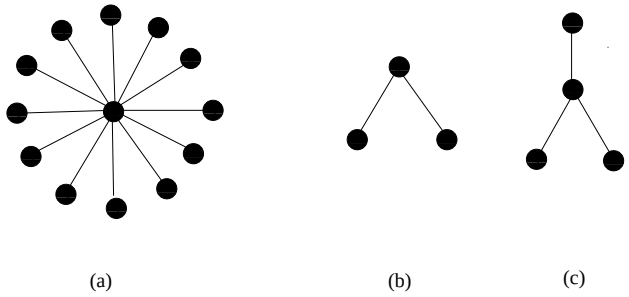

Figure 3.3: Illustrating the bias introduced by the occurrences of hubs (a) on the counts of sub-graphs (b) and (c).

structure and can be applied to study of a number of social and biological networks.

In the previous chapter we discussed on methods of finding the network motifs within a network. But there is major shortcoming of overlapping sub-graphs. As in Figure 3.3 if one hub[1] of degree 12 is present on the network then we will observe 66 sub-graphs of type (b) and 222 of type (c). So this shortcoming is overcome by applying this new method of finding network modules within a network which are non-overlapping. And to find those network modules we discuss an edge betweenness approach.

3.5.1 Hierarchical Clustering

Hierarchical clustering is a traditional method for detecting community structures in networks that work with assigning certain values for every pair of vertices in the network. Here we briefly describe the method.

3.5.1.1 Definition

In hierarchical clustering the data are not partitioned into a particular cluster in a single step. Instead, a series of partitions takes place, which may run from a single cluster containing all objects to n clusters each containing a single object. Hierarchical Clustering is subdivided into agglomerated

[1] A hub is a node with high degree.

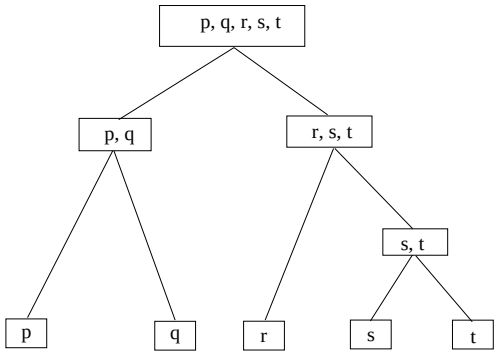

Figure 3.4: Hierarchical clustering dendogram.

methods, which proceed by series of fusions of the n objects into groups, and divisive methods, which separate n objects successively into finer groupings. Agglomerated techniques are more commonly used, and this is the method implemented in XLMiner™. Hierarchical clustering may be represented by a two dimensional diagram known as dendogram which illustrates the fusions or divisions made at each successive stage of analysis. An example of such a dendogram is given in Figure 3.4.

3.5.1.2 Concept

This is a very old method for detecting community structure in networks. The method works as follows,

- First we have to calculate a weight for every pair of vertices in the network. Let it be W_{ij}. It can indicate how closely connected the vertices are.
- Take n vetices with no edges between them.
- Add edges between pairs according to their weights in descending order.
- The resulting graph is a nested set of increasingly large components
- The components can be depicted using a tree.

- Two vertices connected in the lowest level represents their presence in the same community.

3.5.1.3 Weight Function

One possible definition of weight may be *is the number of node-independent paths between vertices*. Two paths that connect the same pair of vertices are said to be node-independent if they share none of the same vertices other than their initial and final vertices. One can similarly also count edge-independent paths. Another possible way to define weights between vertices is to count the total number of paths that run between them (all paths, not just those that are node or edge-independent). long paths contribute exponentially less weight than those that are short. Let A is the adjacency Matrix of the network, and $A_{ij}$is 1 if there is an edge between i and j otherwise 0 then the weights are given by equation 3.2,

$$\mathbf{W} = \sum_{l=0}^{\alpha} (\alpha \mathbf{A})^l = [\mathbf{I} - \alpha \mathbf{A}]^{-1} \tag{3.2}$$

3.5.1.4 Shortcomings

In most of the cases this algorithm gives a reasonable result. In other cases it is less successful. If a vertex is for example connected to the rest of the network by only a single edge then according to this algorithm that single node remain isolated from the network when the communities are constructed. This is a shortcoming of this algorithm.

3.5.2 An Edge Betweenness Approach

To overcome the shortcomings of the hierarchical clustering algorithm an edge betweenness approach was given by Girvan *et al.* [GN02]. A better algorithm for detecting network building blocks in a network was given in [Qia07] by Qiaofeng Yang and Stefano Lonardi. In the following sections we discuss

the working procedure of the algorithm and how it improves computational complexity.

3.5.2.1 Idea of the Algorithm

Girvan *et al.* proposed an algorithm that is based on the following idea,

Instead of trying to construct a measure that tells us which edges are the most central to communities, it focuses on these edges that are least central, the edges that are most "between" communities. The communities are detected by progressively removing edges from the original graph, rather than by adding the strongest edges to an initially empty network. The algorithm's steps for community detection are summarized below

- The betweenness of all existing edges in the network is calculated first.
- The edge with the highest betweenness is removed.
- The betweenness of all edges affected by the removal is recalculated.
- Steps 2 and 3 are repeated until no edges remain.

3.5.2.2 An Improvement and a Parallel Algorithm

Qiaofeng Yang *et al.* worked on Girvan-Newman algorithm.Their work is solely based on 'edge betweenness'. The shortest path of all vertices is calculated. The betweenness of an edge is defined as the number of these shortest paths running through it. The functional modules are loosely connected and they are traversed by the shortest paths between them. If we remove those edges the functional modules are separated out. The outline of the algorithm for counting network module is listed in Algorithm 2.

3.5.2.3 Computational Complexity of the Algorithm

Evaluating the betweenness value for all edges of graph $G = (V, E)$ requires $O(nm)$ time, where$n = |V|$ and $m = |E|$. The iterative removal of all

Algorithm 2 Edge betweenness decomposition algorithm.

Input: Graph G, integer k and a list of all sub-graphs g_i of size smaller or equal to k

Output: Number of correctness of each sub-graph g_i in L

```
C←CONNECTED_COMPONENTS(G)
for each connected component G_d ϵ C do
    ENQUEUE(Q, G_d)
while Q ≠ϕ do
    n, G_c←1, DEQUEUE(Q)
    if NUM_VERTICES(G_c) ≤ k do
            UPDATE_COUNTS(L, G_c)
    else
        while n = 1 do
            e ←EDGE_BETWEENNESS(G_c)
            REMOVE_EDGE(G_c, e)
            C←CONNECTED_COMPONENTS(G_c)
            n ←SIZE(C)
         for each connected component G_d ϵ C do
            ENQUEUE(Q, G_d)
return L
```

$|E|$ edges leads an overall worst case time complexity of $O(mn^2)$ for this algorithm.

3.5.2.4 Comparison Between Two Approaches

Several differences exist between Girvan-Newman and the above algorithm [GN02],

Newman and Girvan's relies on a metric that evaluate the quality of the decomposition, called modularity. In their method, the final decomposition is obtained by "cutting" the dendrogram of the decomposition at the point in which the value of the modularity peaks. On the other hand in their work Qiaofeng Yang *et al.* [Qia07] kept removing edges until the graph disconnects; only if the component is small enough, they stop the process and classify the module in one of 31 non-isomorphic sub-graphs. They referred to their method as *Graph Decomposition Network Module (GDNM)* approach.

3.6 Conclusion

The basic network building blocks of a network are network motifs, network modules etc. In this chapter we have given detail description of both of these building blocks. In the first part of this chapter we have given description of network motifs. Section 3.3 gives brief description of all the previous works being done in the past. In section 3.4 we have given some of the algorithms to find network motifs within a network. Basic technique to find sub-graph, sampling algorithm etc are some of the algorithms being used for this purpose.

In the second part of this chapter we have given description of network module. Hierarchical Clustering, an edge betweenness approach etc are some of the algorithms being discussed in this chapter on network module. There is an improvement on the edge betweenness approach which has also been given in brief.

Chapter 4

Network Modules Using Max Flow-Min Cut Approach

4.1 Introduction

The previous chapters had focused some methods for finding out network motifs or network modules within a network. Some of the techniques had some shortcomings which were also discussed. This chapter mainly focuses on our proposed algorithm Max Flow-Min Cut approach which can be used to find network modules. The Max Flow based method proposed by Flake *et al.* [Flake00, Flake02]. finds an approximate community from a Web graph composed of seed pages and their neighbor pages [Flake00]. This method finds a community within a dense sub-graph. But in our algorithm we find all the k-node network modules within a network repeatedly until all of them are found. This algorithm can be used extensively on biological network. They can be used on other network such as World Wide Web too.

4.2 A Max Flow-Min Cut Based Approach

In this paper we present a graph theoretical algorithm that is based on finding maximum flow in a network. This algorithm works by detecting minimum

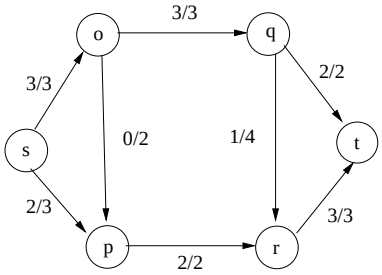

Figure 4.1: Finding minimum cut.

cut in the flow network and separate the desired sub-graph from the network.

4.2.1 Max Flow-Min Cut theorem

The Max Flow-Min Cut theorem is a statement in optimization theory about maximum flows in flow networks. It derives from Menger's theorem. It states that the maximum amount of flow is equal to the capacity of a minimum cut.

The capacity of a cut (S, T) is $c(S, T) = \Sigma_{u \epsilon S, v \epsilon T \mid (u,v) \epsilon T} c(u, v)$.

The following three conditions are equivalent:

1. f is a maximum flow in G.
2. The residual network G_f contains no augmenting paths.
3. $|f| = c(S, T)$ for some cut (S, T).

Given a network with nodes $V = \{s, o, p, q, r, t\}$,and a total flow to the source s to the sink t is 5. which is maximum in this network (see Figure 4.1).

There are three minimum cuts in this network,

Cut	**Capacity**
$S = s, p, T = o, q, r, t$	$c(s, o) + c(p, r) = 3 + 2 = 5$
$S = s, o, p, T = q, r, t$	$c(o, q) + c(p, r) = 3 + 2 = 5$
$S = s, o, p, q, r, T = t$	$c(q, t) + c(r, t) = 2 + 3 = 5$

4.2.2 Ford-Fulkerson Algorithm

This algorithm computes the maximum flow in a flow network. The idea behind the algorithm is to compute the available capacity on all edges in the path, and then send flow along one of these paths. This process is continued until there is no available capacity. A path with available capacity is called an augmenting path. This method was developed by Ford and Fulkerson in 1956 [FF56]. First we look at some definitions,

Residual Network is the collection of edges that can admit more flow. That is, their capacity is not reached by far. Consider a flow network $G(V, E)$. Let one edge (u, v) where $u, v \in V$. If edge (u, v) has capacity $c(u, v)$, the residual capacity of (u, v) can be taken as,

$c_f(u, v) = c(u, v) - f(u, v)$ where $f(u, v)$ is the current flow along edge (u, v). For example, if $c(u, v) = 20$ and $f(u, v) = 10$, then we can increase $f(u, v)$ by $c_f(u, v) = 10$. We can write this idea mathematically as following,

Given a flow network $G = (V, E)$ and a flow f , the residual network of G induced by f is,

$G_f = (V, E_f)$, where, $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$

Given a flow network $G = (V, E)$ and a flow f , an *augmenting path* p is a simple path from s to t in the residual network G_f . By the definition of the residual network, each edge (u, v) on an augmenting path admits some additional positive flow from u to v without violating the capacity constraint on the edge. The shaded path in Figure 4.2(b) is an augmenting path for Figure 4.2(a). Treating the residual network G_f in the figure as a flow network, we can increase the flow through each edge of this path by up to 4 units without violating a capacity constraint, since the smallest residual capacity on this path is $c(a, b) = 4$. We call the maximum amount by which we can increase the flow on each edge in an augmenting path p the residual capacity of p, given by $c_f(p) = min\{c_f(u, v) : (u, v) is\ on\ p\}$.

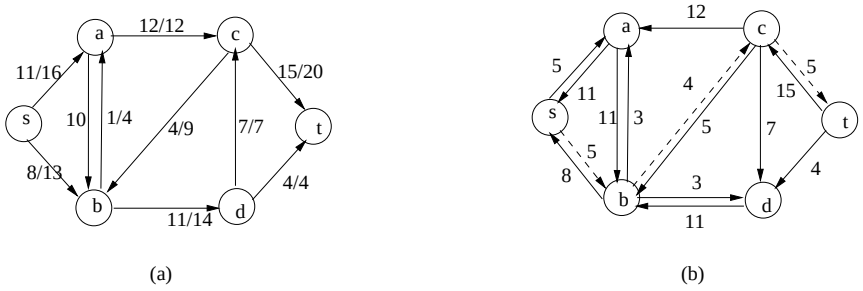

Figure 4.2: a)A sample flow network with flow f, and b) corresponding residual network G_f with augmenting path p shaded; its residual capacity is $c_f(p) = c(a, b) = 4$.

4.2.3 Motivation

This algorithm mainly motivates on finding out all the network modules within a network. It has the additional advantage over the Max Flow based method proposed by Flake *et al.* that it does not stop by only one community. Instead it finds all the communities. And those communities are not overlapping. We can apply this algorithm to biological networks like Protein-Protein Interaction network. The algorithm works on the modified version of flake's algorithm by Asano *et al.* [Asa06].

4.3 A Max Flow-Min Cut Based Algorithm

In this section we outline our proposed Max Flow-Min Cut Based algorithm.

4.3.1 Models

We model the biological network as a flow network. In the flow network, capacity is assigned to each edge according to desired size of the network modules. The modeled graph is then used as an input to our Max Flow-Min Cut based algorithm.

4.3.2 Maximum Flow Network Modules

We recast the problem into a maximum flow framework which analyzes the flow between graph vertices. The $s-t$ maximum flow problem is defined as follows. Given a directed graph $G = (V, E)$, with edge capacities $c(u, v)$ and two vertices, $(s, t) \in V$, find the maximum flow that can be routed from the source, s, to the sink, t, that obeys all capacity constraints. The Max Flow-Min Cut theorem of Ford and Fulkerson proves that the maximum flow of the network is identical to the minimum cut that separates s and t. We choose one or more seed vertices to play the role of the source vertex. Let we define a sink vertex t. then we assign capacity to the edges with the criteria that all edges connecting to the source get capacity of ∞, all edges to the sink get capacity of 1. Other internal edges get capacity of k (the maximum number of nodes in a module). Then we run Max flow - Min cut algorithm to find the maximum flow as well as a minimum cut. We take the portion of the vertex set that contains the source and add the highest degree vertex to the seed set until the size is k. Then we remove the seed set from the graph and run the algorithm recursively thus finding out the later modules. The process is listed in Algorithm 3.

4.4 Results and Discussion

We studied our algorithm on a couple of graphs. The first one to be from a community detection site. The tool is tested with sample graph from a popular community detection tool site[1]. The graph consists of 256 random nodes with 5888 edges. The tool used the algorithm proposed by Reichardt *et al.* [2] . It uses a simulated annealing approach that minimizes the following Hamiltonian

$$\mathcal{H}(\{\sigma\}) = -\sum_{ij}(A_{ij} - \gamma p_{ij})\delta(\sigma_i, \sigma_j)$$

[1] http://projects.forked.de/graph-tool/attachment/wiki/CommunityDetection/community.dot
[2] http://arxiv.org/abs/cond-mat/0603718

Algorithm 3 Algorithm for detecting k-node non-overlapping sub-graphs in networks.

Procedure GET-K-NODE MODULES
Input: Graph G, number of vertices n, number of edges e, size of the non overlapping sub-graph k
Output: A set of k-node disconnected sub-graphs
begin
set G_{new}= G
set $e_{new} = e$
set $n_{new} = n$
while there remains $n_{new} > k$ in graph G
call: G'=DETECT-FLOW-COMMUNITY(G, k)
G_{new}= $G_{new} - G'$
n_{new}= $n_{new} - n'$
$e_{new} = e_{new} - e'$
add G' to the output set
end while
end

Algorithm 4 Algorithm for detecting a k-node sub-graph.

Procedure DETECT-FLOW-COMMUNITY
input: Graph G and a number k
output: a k-node subgroups G' from G
begin
while $|S| <= k$
set S = some random numbered seed vertices $< k$
call: C = EXACT-FLOW-COMMUNITY (G; S; k)
Rank all $V \epsilon C$ by number of edges in C
Add highest ranked non-seed vertices to S
end while
set $G' = S$
output S
end

Algorithm 5 Algorithm for maximum flow minimum cut based partitioning.

Procedure EXACT-FLOW-COMMUNITY
Input: Graph $G(V, E)$, seed vertices set S, number k
Output: A partition based on Maximum Flow-Minimum-cut C such that $S \in C$
begin
    Create artificial vertices, s and t and add to V
    for all $v \in S$ **do**
        Add (s, v) to E with $c(s, v) \equiv \infty$
    end for
    for all $(u, v) \in E$ **do**
        Set $c(u, v) \equiv k$
        if $(v, u) \notin E$ then add (v, u) to$E$ with $c(v, u) \equiv k$
    end for
    for all $v \in V, v \notin S \cup \{s, t\}$ **do**
        Add (v, t) to E with $c(v, t) \equiv 1$
    end for
    call: MAX-FLOW (G, s, t)
end

where p_{ij} is the probability of vetices i and j being connected, which reduces the problem of community detection to finding the ground states of a Potts spin-glass model. Our tool showed similar results according to the output of the tool.

The basic difference was that the simulated annealing algorithm was a non-deterministic one, where as the proposed algorithm is a deterministic one.

The other graph was HPRD protein-protein interaction network of human, including >3600 interactions of >9000 proteins[3].

We have run the algorithm on 10000 interaction edges, and tested for several sub-graph (Numan Protein Protein interaction (PPI) network) collected from *www.hprd.org*.

[3]http://biit.cs.ut.ee/graphweb/welcome.cgi?t=examples
www.hprd.org

4.5 Conclusion

In our algorithm we formulated an way that can detect k-node non overlapping sub-graphs with a Max Flow-Min Cut technique. Though the algorithm depends on initial guesses for seed vertices it provides good results in comparison to the community detection tool we mentioned above. We have also showed a way to deal with biological networks by modeling them as flow networks that can be used to further research on this field.

Chapter 5

Conclusion

In biological networks detecting the densely connected structural elements is always an important subject to study. Structural elements provide and build the very basic features of a biological entity. So studying them is to study the characteristic elements of biological entities. In this thesis paper we tried to investigate the structure of biological networks on their basic structural blocks.

Detecting structural elements is similar to detecting densely connected sub-graphs in a network. Thus the problem of detecting unit blocks in biological networks can be taken as a problem of graph theory. For this reason several researches have been done to deal with the problem from a graph theoretical point of view. Also this problem can also be used to solve community detection in other networks like social networks, data mining, and most importantly the world wide web. From social network part this problem is similar to detecting densely connected groups that can be taken as professional, ethnic, social or other relational connections. For example *Linkedin.com*, groups can be formed of several same minded professional people. So identifying a group from this network is a good way to learn their professional interest as a whole. On the other hand in wide-world web a community can be treated as a collection of links that are accessed on a layered structure like seed pages. Then pages of second layer are accessed from seed pages.

Authors like Flake *et al.* [Flake02, Flake00], Asano *et al.* [Asa06] showed techniques about such community detection in world wide web.

Dealing with biological networks like Protein-Protein interaction (PPI) network is to deal with a related problem of graph theory, where every protein elements can be taken as separate nodes in the graph. So popular graph algorithms can be applied on them. Among many approaches one is detecting networks motifs defined as mostly found sub-graph structures in a network unlike other similar shaped networks. Network motifs exist in all biological networks. Among them most important ones are the feed-forward loop, the single input module and the dense overlapping regulons. U. Alon *et al.* [Alo02]worked extensively on network modules and also built tools that extract those motifs in the form of finding out frequent sub-graphs in a network. Among several methods there are traditional ones that enumerates all frequently occuring sub-graphs in a network which is not computationally effective. An efficient sampling algorithm was proposed by U.Alon *et al.* [Kashtan04]later. It is efficient in the sense that it is not dependent on the size of the network. Another approach was also presented [Koyuturk04] which dealt the problem as a graph data mining problem. Also an extensive work was done by Raghavan *et al.* [Raghban07]. They presented with a nearly linear time localized community detection algorithm based on label propagation. But as for the network motifs they were taken as overlapped densely connected sub-graphs. So overlapping creates some problems specially on network hubs[Qia07]. For this reason an intuitive idea was proposed by Girvan *et al.* [GN02].

This was all about partitioned sub-graphs. Hierarchical clustering algorithms existed several years before, but they were not well accepted. For this reason Girvan *et al.* [GN02] introduced the idea of edge-betweenness that means, finding out and removing the edges, which are said to be the only connection between densely connected sub-graphs. Later these sub-graphs were named as Network Modules [Qia07]. Qiaofeng *et al.* showed an improved algorithm based on shortest path calculations. They also gave a

parallel algorithm based tool that run on multiple processors.

We dealt the problem of finding k-node non overlapping sub-graphs that is based on a maximum flow-minimum cut approach. This algorithm resembles initial work of Flake *et al.* [Flake02]. Though they used it in web communities the authors find it also useful in biological networks. In this method the minimum cut of the network is found out by formulating the problem as a flow network problem. The network building blocks are separated by invoking the minimum cut algorithm. This algorithm was tested with a popular graph detection tool [1]. Also we tested our algorithm on a Human PPI network obtained from *www.hprd.org*. The results were successful in the sense that they provided us with successive k-node structures.

Our method is open for future works like testing it with other graphs and also develop a method of comparison with other techniques rather than flow based.

In conclusion it can be said that finding network building blocks is an important topic to study not only in biological sciences but also in other networks. So from graph theoretical perspective as well as bio-medical science researches can be carried on.

[1] http://projects.forked.de/graph-tool/attachment/wiki/CommunityDetection

Appendix - A

C++ Source Code

The program to find all the network modules within a biological network is given below. The program was written in C++ language.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 100
#define V_SOURCE 0
#define INF 1000
#define N 1000
long int G[N][N], F[N][N];
long int pi[N];
long int CurrentNode[N];
long int queue[N];
long int d[N];
long int numbs[N];
long int edge_array[SIZE][2];
long int n, m, source, sink;
long int edges, vertices;
long int threshold;
long int size_edge;
long int init[2],num[1000];
long int vertexArray[SIZE];
long int com_count=0;
#define oo 1000000000
long int rev_BFS()
{
```

```
    long int i, j, head(0), tail(0);
    for(i = 1; i <= n; i++) numbs[ d[i] = n ] ++;
    numbs[n]–;
    d[sink] = 0;
    numbs[0]++;
    queue[ ++tail ] = sink;
    while( head != tail )
    {
        i = queue[++head];
        for(j = 1; j <= n; j++) {
        if(d[j] < n || G[j][i] == 0) continue;
        queue[ ++tail ] = j;
        numbs[n]–;
        d[j] = d[i] + 1;
        numbs[d[j]]++;
    }
}
return 0;
}
long int Augment()
{
    long int i, j, tmp, width(oo);
    for(i = sink, j = pi[i]; i != source; i = j, j = pi[j]) {
        tmp = G[j][i];
        if(tmp < width) width = tmp;
    }
    for(i = sink, j = pi[i]; i != source; i = j, j = pi[j])
    {
        G[j][i] -= width; F[j][i] += width;
        G[i][j] += width; F[i][j] -= width;
    }
    return width;
}
long int Retreat(int &i)
{
    long int tmp;
    long int j, mind(n-1);
    for(j=1; j <= n; j++)
```

```
    {
        if(G[i][j] > 0 && d[j] < mind)
        mind = d[j];
    }
    tmp = d[i];
    numbs[d[i]]–;
    d[i] = 1 + mind;
    numbs[d[i]]++;
    if( i != source ) i = pi[i];
    return numbs[ tmp ];
}
// Main procedure
long int find_max_flow()
{
    long int flow(0), i, j;
    rev_BFS();
    for(i=1; i<=n; i++) CurrentNode[i] = 1;
    i = source;
    for( ; d[source] < n ; ) {
    for(j = CurrentNode[i]; j <= n; j++)
        if( G[i][j] > 0 && d[i] == d[j] + 1 ) break;
    if( j <= n )
    {
        CurrentNode[i] = j;
        pi[j] = i;
        i = j;
        if( i == sink ) { flow += Augment(); i = source; }
    }
    else { CurrentNode[i] = 1; Retreat(i); }
    }
    return flow;
}
long int minimum_cut()
{
    long int i;
    find_max_flow();
    for(i=0;i<n;i++)
    for(i=0;i<=n;i++)
```

```
    return 0;
}
long int getMax()
{
    long int max=-1;
    long int max_num=-1;
    long int pos=0;
for(long int i=0;i<n;i++)
    {
        if(d[i]>max)
        {
        max=d[i];
        max_num=i; //d[i]=-2;
        }
    }
    if(max_num!=-1)
    {
        d[max_num]=-2;
    }
    return vertexArray[max_num];
}
bool check(long int x, long int a[],long int size)
{
    for(long int j=0;j<=size;j++){
        for(long int i=0;i<edges+2;i++)
        {
            if((edge_array[i][0]==a[j] && edge_array[i][1]==x)||(edge_array[i][0]==x
&& edge_array[i][1]==a[j]))
            { return true; }
        }
    return false;
}
void remove_old()
{
    long int temp_edge[10000][2];
    long int temp_vertex[10000];
    long int count=0;
    long int count2=0;
```

```
bool found=false;
for(long int i=0;i<edges;i++)
{
    for(long int j=0;j<threshold;j++)
    {
        if((edge_array[i][0]==num[j])||(edge_array[i][1]==num[j]))
        {
            found=true; break;
        }
        else { found=false; }
    }
    if(!found)
    {
        temp_edge[count][0]=edge_array[i][0];
        temp_edge[count][1]=edge_array[i][1];
        count++;
        found=false;
    }
}
for( i=0;i<vertices;i++)
{
    for(long int j=0;j<threshold;j++)
    {
        if(vertexArray[i]==num[j])
        {
            found=true;
            break;
        }
        else { found=false; }
    }
    if(!found)
    {
        temp_vertex[count2]=vertexArray[i];
        count2++;
        found=false;
    }
}
edges=count;
```

```
    for(i=0;i<count;i++)
    {
        edge_array[i][0]=temp_edge[i][0];
        edge_array[i][1]=temp_edge[i][1];
    }
    vertices=count2;
    for(i=0;i<count2;i++)
    {
        vertexArray[i]=temp_vertex[i];
    }
}
void clear()
{
    for(long int i=0;i<N;i++)
    {
        for(long int j=0;j<N;j++)
        { G[i][j]=0; }
    }
    for(i=0;i<N;i++)
    { d[i]=0; }
}
void find_community()
{
    long int size=edges;
    long int i=edges;
    com_count++;
    edge_array[i][0]=V_SOURCE;
    edge_array[i][1]=init[0];
    i++;
    edge_array[i][0]=V_SOURCE;
    edge_array[i][1]=init[1];
    for(long int j=0;j<vertices;j++)
    {
        i++;
        edge_array[i][0]=j;
        edge_array[i][1]=vertices+1;
    }
    size_edge=i;
```

```
long int a;
long int b;
clear();
for(j=0;j<=i;j++)
{
    if(edge_array[j][0]==0)
    {
        a=edge_array[j][0];
        b=edge_array[j][1];maximum
        G[a][b]=INF;
    }
    else if(edge_array[j][1]==vertices+1)
    {
        a=edge_array[j][0];
        b=edge_array[j][1];
        G[a][b]=1;
    }else
    {
        a=edge_array[j][0];
        b=edge_array[j][1];
        G[a][b]=threshold;
    }
}
n=vertices+2;
m=edges+2+vertices;
sink=vertices+1;
source=0;
minimum_cut();
printf("getting community %d: \n",com_count);
long int com=0; //threshold;
long int temp=0;
for(i=0;i<n;i++)
{
    temp=getMax();
    if(i>0)
    {
        if(check(temp,num,com))
```

```
            {
                if(temp!=V_SOURCE&&(temp!=vertices+1))
                {
                    printf(" %d",temp);
                    num[com]=temp;
                    com++;
                }
            }
            if(com>=threshold)break;
        }
        else if(i==0)
        {
            if(temp!=V_SOURCE&&(temp!=vertices+1))
            {
                printf(" %d",temp);
                num[com]=temp;
                com++;
            }
            if(com>=threshold)break;
        }
    printf("\n");
    if(vertices>threshold)
    {
        remove_old();
        printf("\n");
        find_community();
    }
}
void main()
{
    long int i=0;
    freopen("b.in","rt",stdin);
   printf("Enter vetices no, edges no, threshold\n"); scanf("%d%d%d",&vertices,&edges,&threshold);
    long int source=rand()%edges;
    source=(vertices%threshold);
    for(int k=1;k<=vertices;k++)
    {
        vertexArray[k]=k;
```

```
    }
    while(i<edges)
    {
        scanf("%d%d",&edge_array[i][0],&edge_array[i][1]);
        if(i==0)
        {
            init[0]=edge_array[i][0];
            init[1]=edge_array[i][1]; }
        i++;
    }
    find_community();
}
```

Bibliography

[Alo02] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, U. Alon, *Network Motifs: Simple Building Blocks of Complex Networks*, Science 25, Vol. 298. no. 5594, pp. 824 - 827, 2002.

[Asa06] Y. Asano, T. Nishizeki and M. Toyoda, *Mining Communities on the Web Using a Max-Flow and a Site-Oriented Framework*, IEICE, Vol E89-D, pp. 2606-2615, 2006.

[Babu04] M. M. Babu, N. M. Luscombe, L. Aravind, M. Gerstein, and S. A. Teichmann, *Structure and evolution of transcriptional regulatory networks.* Curr Opin Struct Biol 14(3), pp. 283-291, 2004.

[Conant03] G. C. Conant, and A. Wagner, *Convergent evolution of gene circuits.* Nat Genet 34(3), pp. 264-266, 2003.

[Dekel05] Dekel, E. and U. Alon, *Optimality and evolutionary tuning of the expression level of a protein.* Nature 436, pp. 588-592, 2005.

[FF56] L. R. Ford, D. R. Fulkerson, *Maximal flow through a network.* Canadian Journal of Mathematics 8, pp. 399–404, 1956.

[Flake00] G. W. Flake, S. Lawrence, C. L. Giles, *Efficient Identification of Web Communities*, Sixth ACM SIGKDD international conference ,pp. 150 - 160, 2000.

[Flake02] G. W. Flake, S. Lawrence, C. L. Giles, F. M. Coetzee, *Self-Organization and Identification of Web Communities.* IEEE Computer 35(3), pp. 66–71, 2002.

[GN02] M. Girvan, M. E. J. Newman, *Community structure in social and biological networks,* Proceedings of the National Academy of Sciences, Vol. 99. no. 12, pp. 7821-7826, 2002.

[Kashtan04] N. Kashtan, S. Itzkovitz, R. Milo and U. Alon. *Efficient sampling algorithm for estimating sub-graph concentrations and detecting network motifs.* Bio-informatics 20, pp. 1740-1758, 2004.

[Koyuturk04] M. Koyuturk, A. Grama, and W. Szpankowski, *An efficient algorithm for detecting frequent subgraphs in biological networks.* Bioinformatics 20, pp. i200–i207, 2004.

[Maslov02] S. Maslov and K. Sneppen, *Specificity and stability in topology of protein networks.* Science 296, pp. 910–913, 2002.

[Milo02] S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon, *Network motifs in the transcriptional regulation network of escherichia coli.* Nature Genetics 31, pp. 64–68, 2002.

[Qia07] Q. Yang, S. Lonardi, *A Decomposition Approach for Discovering Network Building Blocks*, BIOKDD'07, pp. 50-59, 2007.

[Raghban07] Usha Nandini Raghavan, Reka Albert and Soundar Kumara. *Near linear time algorithm to detect community structures in large-scale networks.* Physical Review E 76, 2007.

[Sattath04] E. Yeger-Lotem, S. Sattath, N. Kashtan, S. Itzkovitz, R. Milo, R. Y. Pinter, U. Alon, and H. Margalit, *Network motifs in integrated cellular networks of transcription-regulation and protein-protein interaction.* PNAS 101, pp. 5934–5939, 2004.

[Yang07] F. Luo, Y. Yang, C. F. Chen, R. Chang, J. Zhou, and R. H. Scheuermann, *Modular organization of protein interaction networks.* Bioinformatics 23, pp. 207–214, 2007.

Index