UNIVERSITY OF CINCINNATI

_____________ , 20 _____

I,____,
hereby submit this as part of the requirements for the degree of:

____

in:

____

It is entitled:

____

____

____

____

Approved by:

____

____

____

____

____

SHORTCUT BASED GRAPH COARSENING FOR PROTEIN INTERACTION NETWORK VISUALIZATION

A thesis submitted to the
Division of Graduate Studies and Research
of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in the Department of
Electrical and Computer Engineering and Computer Science
of the College of Engineering

July, 2001

by

Li Zhong

B.E., Zhejiang University, 1992
M.S., Dalian Institute of Chemical & Physics, Chinese Academy of Sciences, 1995

Committee Chair: Dr. Yizong Cheng

Abstract

Protein-protein interactions play important role in various biological processes. These interactions inside an organism constitute a complex network. A global view of this network is very useful in providing molecular and genetic scientists with a reference guide to aid detailed exploration of the functions of proteins. In this thesis, we build up a network, named as CID(Connected Interaction Database), whose links are defined as the interactions between proteins in *S.Cerevisiae*. The topological properties of this CID network have been studied and compared with another network called CSD (Connected Similarity Database), in which a link denotes either a physical interaction or functional similarity. The topological results imply that both CID and CSD possess the “small world” structure, where shortcuts play important role in shortening the path length between different clusters. We propose a graph-coarsening algorithm based on these shortcuts and cut nodes to identify clusters presented in these network and generate a simplified “backbone” structure of CID and CSD. Studies on the “functional” distribution inside some clusters demonstrate that proteins with similar function (in broad sense as referring to both “cellular role” and “biological function”) are more likely to be clustered together. Using the combination of physical interaction and sequence similarity data, we correctly predicted functional categories for 82.67% among the 3,116 characterized proteins with at least one partner (the proteins that have connections with the protein under study) of known function and functional categories for 907 previously uncharacterized proteins have been predicted.

AKNOWLEDGEMENTS

I wish to thank my research advisor, Dr. Yizong Cheng, for his guidance, encouragement, understanding, and patience throughout this work. He is such a great person to be around, and his advice and encouragement have been invaluable.

I would also like to thank my committee members, Dr. Fred Annexstein, Dr. John Schlipf and Dr. Kenneth A. Berman for their valuable advice and helpful discussions.

My sincere thanks are given to my husband Ruzhi Zhang, daughter Catherine and other family members. Their unconditional love supported me throughout the completion of this work.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Protein-protein interactions play important role in various biological processes such as the regulation of cellular pathway, the formation of enzyme complexes, *etc*. With the availability of complete genome sequences and the development of high-throughput two-hybrid system, large-scale identification and characterization of these interactions have been performed in several research groups[1]. In particular, yeast *Saccharomyces Cerevisiae* has received extensive study since its genome was sequenced in 1996[2]. Recently, the Fields group at Howard Hughes Medical Institute analyzed published data on more than 2,709 interactions involving 2,039 different yeast proteins and assembled the first map of yeast protein interaction network[1].

Even though this protein interaction map might be incomplete or even contain some error links due to limited information available for this huge protein database, it is still very useful in providing molecular and genetic scientists with a reference guide to aid detailed exploration of the functions of yeast proteins. Using this map, Fields group correctly predicts the functional category for 72% of the 1,393 characterized proteins based on the known functions of its interacting partners. In addition to protein function prediction, this database of interactions has also been combined with DNA microarrays and quantitative proteomics to analyze the perturbations to critical pathway components in a cellular pathway model[3]. More than that, study of protein under global proteins interaction network has revealed more unidentified information such as the relationship between lethality of a protein and its centrality in the protein interaction networks[4]. Based on the above observation, one objective of this thesis is to understand this interaction map from a large-scale perspective and explore the relationship between the underlying topological connectivity and the biological structure of this interaction network. We anticipate that such study might provide us more insights into the biological process that generated such network.

The thesis will start with some basic introduction about the major terminology used, followed by an overview of the source data used in our protein-protein interaction network. Then, we explore this "gold mine" from a topological view covered in Chapter 2. Next, a graph coarsening algorithm based on the "small world" connectivity presented in this network is proposed in Chapter 3, ending with a "coarsened" globular graph for this network. The biological relationship inherent in some clusters identified has been studied. In this thesis, we build a new version of protein connectivity network by considering not only the interactions between proteins, but also the sequence similarity. In Chapter 4, we report a more accurate result of functional predication from this new map. The topological properties of this new network are also studied in Chapter 2. Finally, the summary of results in the thesis is presented in Chapter 5.

Chapter 2

Topological Study

2.1 Graph Theory

Many systems in nature, from the neural networks in the biological world, the social structure in our communities, to the Internet and World Wide Web, can be effectively represented by a graph $G(V, E)$ where V(G) is the set of vertices $V(G)= \{v| v\in G\}$ and E(G) consists of edges $E(G) =\{e | e\in G\}$. In such a graph, each vertex *v* represents a single element of the system (a neuron in the brain, a person in our society, or a web page on the WWW), while an edges *e* is the links or connections(neuron interaction, friendship, or web page hyperlinks) between the vertices. Throughout this thesis, we may use *vertex* or *node* interchangeably to represent a single element in the network. We also refer to the number of vertices in G as |V| and the number of edges in G as |E|. All the graphs under study are considered as undirected and unweighted.

In order to understand the dynamic behavior of a natural system, it is very important to explore the topological properties of the underlying network, which is a major research area in graph theory. Many terms have been introduced to facilitate the study. In this thesis, the following terminology will be frequently used in our discussion.

1) *Degree*: The number of edges incident on a given vertex ν, i.e., the size of the adjacent list of ν, is denoted as $k_ν$. The average degree k of a graph G refers to the number averaged over G.

2) *Leaf node:* A vertex μ with $k_μ = 1$ is called a **leaf node**. One example is node μ in Figure 2.1

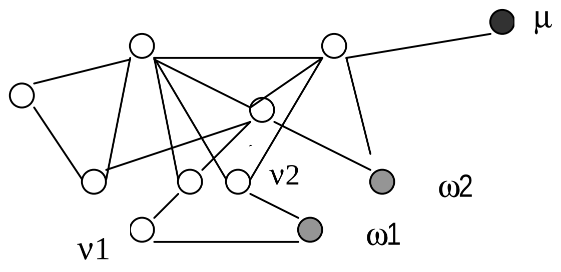

Figure 2. 1 μ is a leaf node in this graph, and ω1 and ω2 are the minor nodes.

3) *Minor node:* As shown in Figure 2.1, vertex ω1 has only two connections with ν1 and ν2, even though the removal of edges (ω1,ν2) or (ω1,ν1) might bring some changes to the overall graph connectivity, removal of vertex ω1 and merge edge (ω1,ν2) and (ω1,ν1) into edge (ν1, ν2) will not affect other vertices. Under such condition, we call ω1 a **minor node**.

4) *Shortcut(r):* For an edge (μ,ν)∈ E(G), where μ ∈V(G), ν∈ V(G) and neither μ nor ν is a leaf node or minor node, if the shortest distance between vertex μ and vertex ν is larger than r at the absence of edge (μ,ν), then edge(μ,ν)is a **shortcut** of graph G(V,E) at level r (r ≥ 2).

5) *Distance d:* Let μ ∈V(G), ν∈ V(G), the distance d(μ,ν) in G is defined as the shortest path length between node μ and ν. For the undirected, unweighted graph, d(μ,ν) equals to the minimum number of links between μ and ν, which can be

obtained from the level of one node in the breadth first search tree rooted at another node.

6) *Subgraph S(G):* Let S be a graph S(V′, E′), if V′ (S) $\subseteq$ V(G) and E′ (S) $\subseteq$ E(G), then S(V′, E′) is called a subgraph of G(V,E).

7) *Subgraph neighborhood $\Gamma(S)$:* Let S be a subgraph of G, the neighborhood $\Gamma(S)$ in G is the subgraph that consists of all vertices adjacent to any of the vertices in S, but not including any vertex of S. $\Gamma(S)$ maintains all links connecting among these vertices in the original graph G.

8) *ith Neighborhood of vertex $\Gamma^i(\nu)$:* Let the 1st neighborhood $\Gamma^1(\nu)$ of a vertex $\nu \in V(G)$ be the subgraph of G which consists of the vertices adjacent to ν (not including ν itself). Then the ith neighborhood $\Gamma^i(\nu)$ of a vertex $\nu \in V(G)$ is defined as :

   $\Gamma^i(\nu) = \Gamma(\Gamma^{i-1}(\nu)) + \Gamma^{i-1}(\nu)$ ($i > 1$)

   In an unweighted and undirected graph, $\Gamma^i(\nu)$ can be viewed as the subgraph S(V′, E′) where V′ (S) = $\{ \mu \mid \mu \in V(G), \mu \neq \nu,$ and distance $d(\mu, \nu) \leq i\}$.

9) *Cut node:* For a vertex $\nu \in V$ (G), let S be the set of vertices that adjacent to ν, *i.e.* S = $\Gamma^1(\nu)$. If the total vertices in S is larger than 4, and there exist at least 2 pairs of nodes in S (μ,ω), (μ',ω') such that $d(\mu,\omega)$ and $d(\mu',\omega')$ is larger than r (r >2) at the absence of ν and its corresponding edges in G, then μ is called a **cut node** of graph G(V, E) at level r.

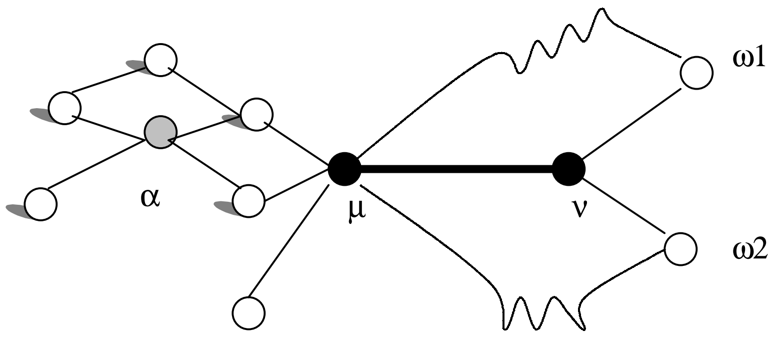

Figure 2. 2 Shortcut (μ ,ν) at r implies that node μ and ν are cut nodes at level > r-1

In fact, if an edge (μ,ν) is a shortcut at level r of G and r ≥2, then both μ and ν are **cut nodes** of G at level > r-1. This can be proved as following:

> According to the definition of shortcuts, both μ and ν have at least three adjacent nodes. Let assume ω1 and ω2 be the adjacent nodes of ν, then d(μ, ω1) and d(μ ,ω2) must be larger than (r-1). Otherwise the path μ →ω1→ν, μ →ω2→ν will make *d(μ ,ν)* ≤ r in the absence of edge (μ ,ν), as shown in Figure 2.2, and contradict with the premise that (μ ,ν) is a shortcut at level r.

10) *Clustering Coefficient* $\gamma^i(\nu)$ *and* $\gamma^i(G)$*:* Let Kv be the number of vertices in ν's ith neighborhood *Γ* ^{i}*(*ν*)*. Then at most $\binom{Kv}{2}$ edges can exist between them. The fraction of these possible edges that actually exist is called **clustering coefficient** $\gamma^i(\nu)$. More precisely,

$$\gamma^i(v) = \frac{|E(\Gamma^i(v))|}{\binom{Kv}{2}}$$

Where $|E(\Gamma^i(v))|$ is the number of edges in $\Gamma^i(v)$, and $\binom{Kv}{2}$ is the total number of possible edges in $\Gamma^i(v)$.

The clustering coefficient $\gamma^i(G)$ of graph G refers to the number averaged over all $\nu \in V(G)$, i.e. $\gamma^i(G) = \frac{\sum_{v \in V(G)} \gamma^i(v)}{|V(G)|}$ Thus, $\gamma^1(G) = 1$ for a connected graph would imply *G* is a complete connected graph, where every vertex $\nu \in V(G)$ is connected with other vertices. Ideally clustering coefficient would represent the local "neighboring" linkage information, *i.e.* how likely two people with a common friend would know each other in a social network. Higher $\gamma^i(G)$ value implies that such probability is larger. Considering this, when we calculate the clustering coefficient of overall graph $\gamma^i(G)$, we normally don't count those leaf nodes with single connections.

11) *Characteristic Path length L(G):* Let $d(\mu,\nu)$ be the shortest distance between vertex μ and ν, the characteristic path length of graph *L(G)* is defined as the average over all pairs of vertices, i.e. $L(G) = \frac{\sum_{\mu,\nu \in V(G), \mu \neq \nu} d(\mu,\nu)}{\binom{|V|}{2}}$

2.2 Small World Phenomenon

When a network involves thousands of nodes and connections between them, the whole system structure becomes very complex. Three different models have been proposed to study these complex networks:

1) Random network. This classical theory is first introduced by Erdös & Renyi[5]. It assumes that each node chooses another to make a connection randomly with probability *p*. Thus, the whole network is statistically homogeneous such that most of the nodes have the same number of links, implying that the probability of finding a node with link number > k decays exponentially (P(#of links > k) = e^{-k}).

2) Regular network. The connection topology is regular if each node is connected only to its immediate neighbors in a regular lattice.

3) "Small World" network. Many networks in the real world lies somewhere between the above two extremes. Watts and Strogatz[6] have demonstrated that if a regular lattice is "rewired" by introducing certain degree of disorder, at some point, the network will keep the highly clustered structure like regular graph, while the minimum distance between any two randomly chosen nodes has been shortened to that of a random graph. The name of *"small world"* is given to such a highly clustered network with small global separation since it is very similar to our life experience of running into a complete stranger at a party and finding mutual friends after a short conversation. This is known as "six degrees of separation".

So far, studies show that several man-made or real world networks exhibit this *small world* phenomenon, including the World Wide Web[7], the power grid of the western states, the collaboration graph of actors and the neuron network of *C. elegans*[2,8] etc. Actually, one good example of small world topology can be seen from the "Kevin

Bacon" game, in which it is very difficult to find any actor whose degree of separation from Kevin Bacon (KB) is greater than 4 if we define one separation as direct collaboration with KB, second separation as collaboration with one of KB's direct collaborators, and so on.

This small world topology will strongly affect the dynamics properties of the network. Watt and Strogatz have demonstrated theoretically that a small faction of the random links added to a regular lattice graph will speed up the propagation of a disease across the whole graph. This suggests that a few people traveling from place to place, or from one social community to another, might be enough to trigger a full-blown epidemic. On the other hand, such a small world topology can also be used in a positive way. For example, Adamic[9] demonstrated that the pages corresponding to a particular search query are more likely than not to form a small world network. Therefore, a "smart" search engine that presents just the center of each "cluster" instead of lists of many sequential entries from the same sites and sorts these centers according to the sizes of "clusters" will give the user a brief and expandable search results.

Before Watt and Strogatz's formal mathematical modeling and proposal of this "small world" phenomenon, people often divided complex networks into to two major classes according to the connectivity distribution P(k), which is the probability that a node is connected with k other nodes in such a network. Two kinds of distribution exist in complex networks:

1) Homogeneous networks, in which *P(k)* follows the Poisson distribution and each node has approximately the same number of connections, $k \approx \langle k \rangle$, where $\langle k \rangle$ is

the statistical average of link numbers. Thus, the probability to find a highly connected node decays exponentially, *i.e.* $P(k) \sim e^{-k}$ for ($k >> <k>$). Erdös's random graph is such a case.

2) Heterogeneous networks, is also referred as *scale-free* networks, for which $P(k)$ decays under a power law, i.e. $P(k) \sim k^{-\gamma}$. Under such distribution, some highly connected nodes are statistically significant rather than being prohibited in homogeneous network. Some examples include WWW and the internet, with observed γ between 2 ~3. [10] One important consequence of such network is the high degree of tolerance against errors and vulnerability towards attack.

In our opinion, *scale-free* network is a special case of "small world" instead of an exclusive category outside, since "small world" should be interpreted in a very broad sense as denoting those network topologies with locally dense areas connected with some "long jumps" to reach the outside world. In the following discussion, we will see a network displaying both power law properties and "small worldness".

2.3 Source Data

In this thesis, we mainly study two network structures. The first one we used involves 1,825 individual proteins and the 2238 links that are defined as the identified physical interactions between proteins. These interaction data are derived from several combined and non-overlapping sources such as the databases of MIPS[11], DIP[12], and systematical two-hybrid analyses[13] [14]. These combined interaction data are also downloadable at http://depts.washington.edu/sfields/yplm/data/NB_data.html. We will refer to this pure interaction network as **PID**(physical interaction database) in our following discussion.

However, **PID** is not a connected graph. Through *Breadth First Search*, this network was found to consist of 186 individual components. The largest connected component has total 1297 proteins and 1862 interactions between members. The second largest connected component has 11 proteins, and the 95% of the other components have only two proteins. We denote the largest connected component as **CID**.

In the second network studied, we define a link between two proteins as not only a physical interaction, but also a sequence similarity (homologies or orthologies). We retrieved such data from the YPD™ and obtained total 24,647 distinct links between 6,108 proteins from the January, 2001 version. Similar to **PID**, this original similar-sequence-and-interaction protein linkage graph (referred as **SID**) is also unconnected and consists of 280 individual components by performing *Breadth First Search*. The largest connected component has total 3725 vertices and 23,986 edges in it. All other component are quite trivial with only 2~10 vertices each. The largest connected component is singled out in our study and abbreviated as **CSD.**

2.4 Results and Discussion

During our study, we make the following assumption for both networks:

1. All nodes (representing each yeast protein) are treated as identical, featureless vertices.
2. All interactions / relationships are assumed to be bi-directional; hence the resulting graphs are undirected.
3. All interactions are assumed to be identical, ignoring the fact that some of the interactions might be much closer than others. Thus, we get unweighted graphs.

2.4.1 CSD Network

For an unweighted and undirected graph, the characteristic path length can be obtained by performing a breadth first search rooted at every possible node and averaging over all the paths. The resulting *L(G)* of 4.974 is short, comparing to the magnitude of the number of vertices involved and the diameter of 16. The histogram of the shortest distance between two nodes *d(i, j),* shown in Figure 2.3, indicates that more than 50% of the *d(i, j)s* fall into the category of 4~5. This is not a surprising fact if we consider the requirements in the biological world that each protein must communicate/interact with others effectively to accomplish certain function and adapt to the changes of environment.

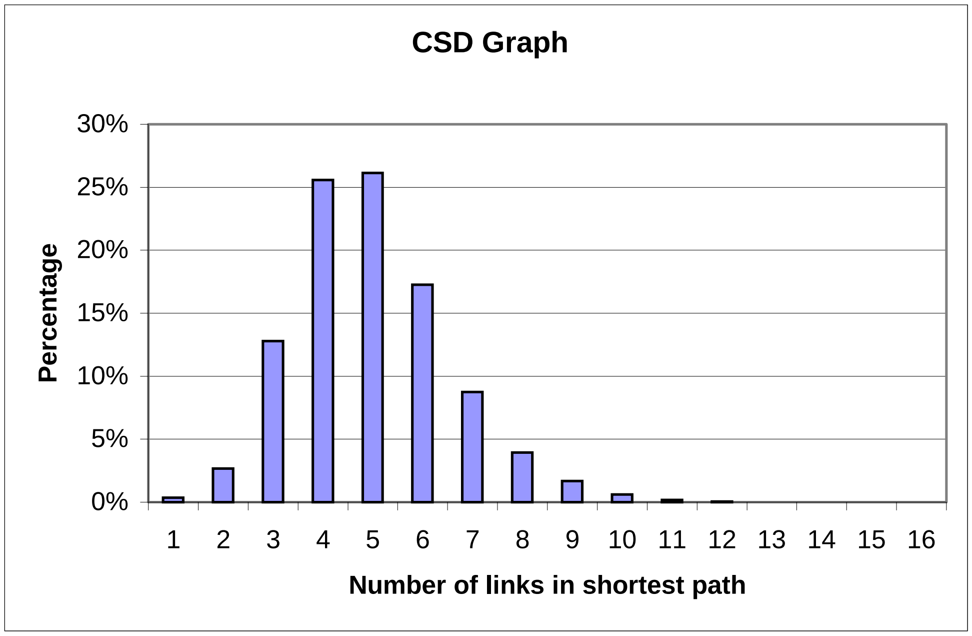

Figure 2. 3 Histogram of the shortest distance between every possible pair of nodes *d(i, j)* in CSD graph

On the other hand, the local area around most of the vertices in CSD graph is very dense, as indicated by their clustering coefficient. Figure 2.4 is the clustering coefficient $\gamma^i(\nu)$ (where i = 1) histogram for this CSD graph. It shows that about 17% of the vertices have $\gamma^i(\nu)$ =1.0, which means all neighbors of vertex ν are also connected with each other. The whole graph clustering coefficient average over all vertices equals to 0.590. Leaf nodes are not counted when computing $\gamma^i(G)$ (i =1).

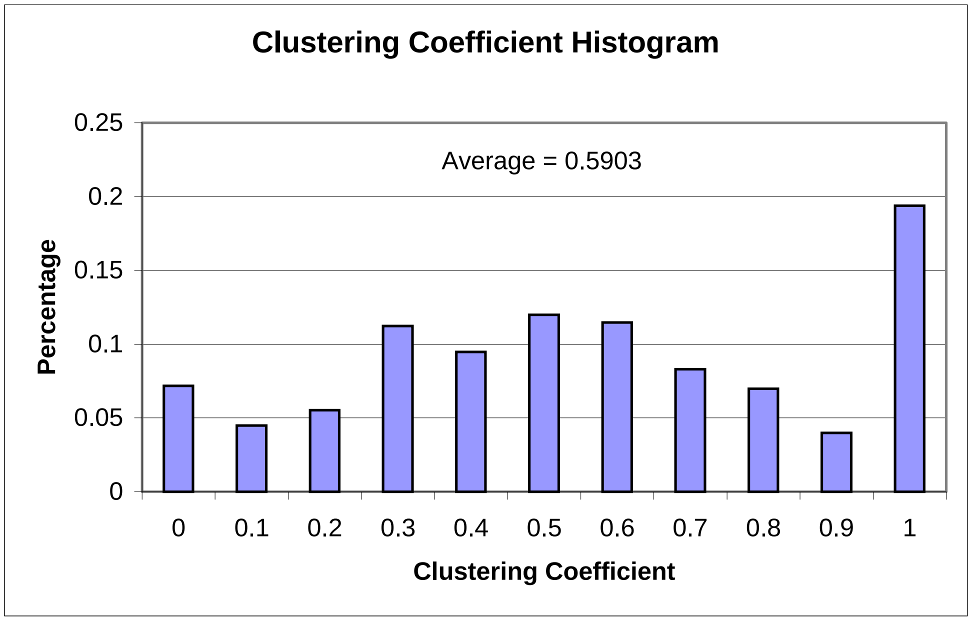

Figure 2. 4 Clustering Coefficient Distribution of CSD Graph (level i =1)

With the short characteristic path length L(G) and high local density $\gamma(G)$, this CSD graph possesses the so-called "small world" connectivity. In this "small world", most of the nodes have their own highly clustered local areas so that each can reach all other nodes in the same local area within 1~2 local edges through multiple paths. In order to connect one cluster to another to achieve shorter path length between any two nodes in the graph, some "global" edges (shortcuts) are required. Therefore, shortcuts play a very import role in the global connection.

As shown in Figure 2.5, the probability *P(k)* that a given yeast protein interacts with k other yeast proteins follows a power law. *P(k)* is computed as the percentage of vertices with k links over |V|. Previous study shows that this power-law topology is also shared by the protein-protein interaction network of the bacterium *Helicobacter Pylor*[15].

One characteristic of the power law network is that it is a highly nonhomogeneous scale-free network in which a few highly connected proteins play a central role in mediating connections among numerous, less connected proteins. This can be demonstrated by the fact that many highly connected proteins are either involved in "shortcut" or cut nodes, which will be discussed in the next chapter.

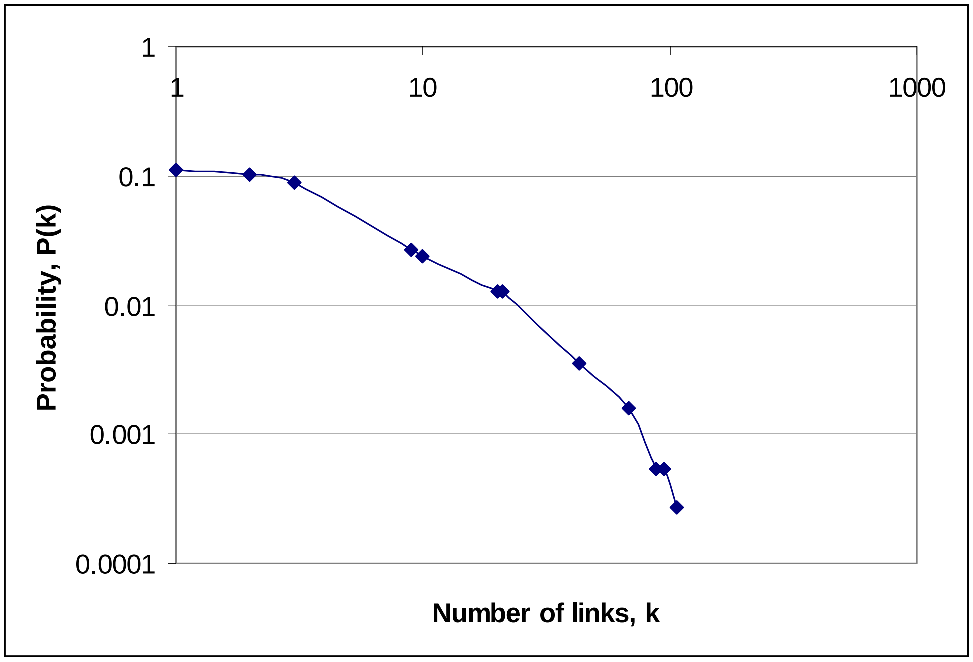

Figure 2. 5 Connectivity distribution P(k) in CSD graph.

***P(k)* = the probability that a node has k links.**

2.4.2 CID Network

For the CID network, the connection is much more sparse than CSD with degree k ~2.86 and half of the vertices out of 1,267 dominated with less or equal than 2 links. By making similar path length and clustering analysis, we obtain a characteristic path length *L(G)* of 7.63 and a much lower clustering coefficient $\gamma^1(G)$ of 0.217, comparing to 0.59 in CSD. This $\gamma^1(G)$ of 0.217 is still much higher than that of 0.08 in a random graph with same number of vertex and edges. It should be kept in mind that we don't count those single nodes in the calculation of $\gamma^1(G)$. Therefore, $\gamma^1(G)$ should be a reasonable indication of the connectedness inside a "close neighboring" clique if one exists. In addition, our further calculation towards "neighbor's neighbor", i.e. $\gamma^2(G)$, leads to a higher value of 0.312 than $\gamma^1(G)$ of 0.217. Such tendency is also indicated in the histogram of $\gamma^1(G)$ and $\gamma^2(G)$ in Figure 2.6, with more nodes displaying some degree of clustering instead of complete separation. Such a tendency is unlikely to happen in a random, homogenous network. We will postpone our explanation until next section when we compare the graphical structure difference between CSD and CID network. The point we want to make here is that the CID network is more likely to be a "small world" rather than a random graph even though its clustering properties is not as obvious as CSD graph and some verified cases.

Figure 2.7 is the plot of probability P(k) versus number of links k in CID graph. The straight line under logarithms coordinate indicates that CID graph is a *scale-free* network. This may further distinguishes this CID graph from a random graph.

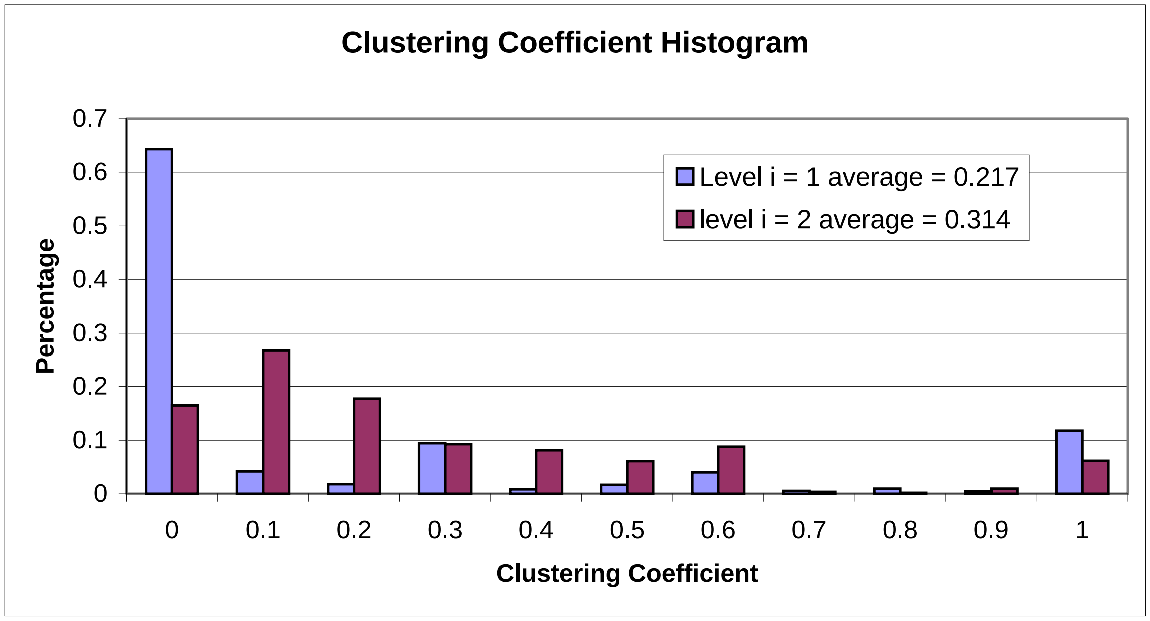

Figure 2. 6 Clustering Coefficient Distribution of CID Graph.

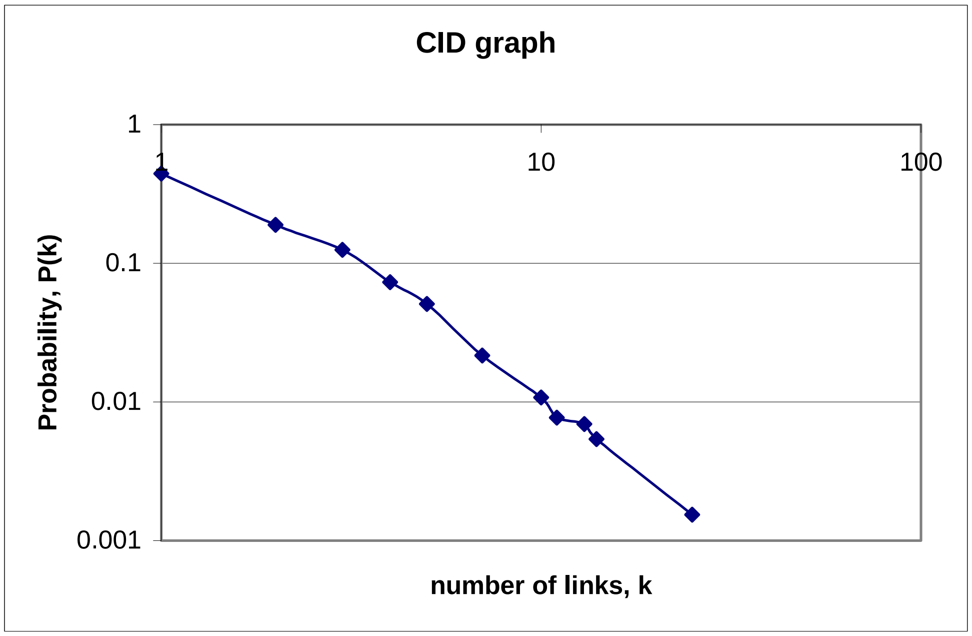

Figure 2. 7 Connectivity distribution P(k) in CID graph.

***P(k)* = the probability that a node has k links.**

2.4.3 Comparison between CSD and CID

The basic parameters for both CID and CSD graph are summarized in Table 2.1.

Even though the total node number in CID network is only 1/3 that of the CSD network, both the path length and diameter are higher than those of CSD network. If we denote Φ as the percentage of shortcut(r) over all the edges in graph, as shown in Table 2.1 and Figure 2.9, Φ in CSD is much lower than those in CID graph with similar decaying tendency. (The detail of how to calculate shortcut is covered in the next chapter). At first glance, the larger percent of shortcut presented in CID with fewer nodes than CSD should result in a shorter overall path length since shortcut plays an important role in contracting the path length between nodes in different clusters and decreasing characteristic path length of graph. This conjecture contradicts the real results. Our explanation is that the clusters of CID have a much smaller size than those of CSD, causing the total cluster number comparable to CSD. On the other hand, CID has much looser cluster structure and thus longer path length inside a cluster than CSD. As mentioned earlier, CID graph is much sparser than CSD in the sense of both lower degree of connection and lower clustering coefficient. However, the plot of γ at different neighboring level between 1 and 5, as illustrated in Figure 2.8, indicates that the there is a peak in the CID plot at level 2 instead of 1, which indicate that the nodes in CID is more likely to connect with its neighbor’s neighbor rather than make a triangular link as in CSD graph. In addition, there exist a significant number of **cut nodes** in the CSD graph.

In our graph coarsening process covered in Chapter 3, we will show that the existence of these cut nodes will make the path length shorter even without shortcut.

Table 2. 1 CID and CSD Graph Basic Parameters

Parameter	CSD	CID
Total # of vertices |V|	3,725	1,297
Total # of edges |E|	23,986	1,862
Degree k	12.88	2.86
Average Clustering Coefficient	0.590	0.217
Diameter D(G)	16	23
Characteristic Path Length L(G)	4.974	7.627
Percentage of shortcut Φ (level =2)	0.0374	0.4557

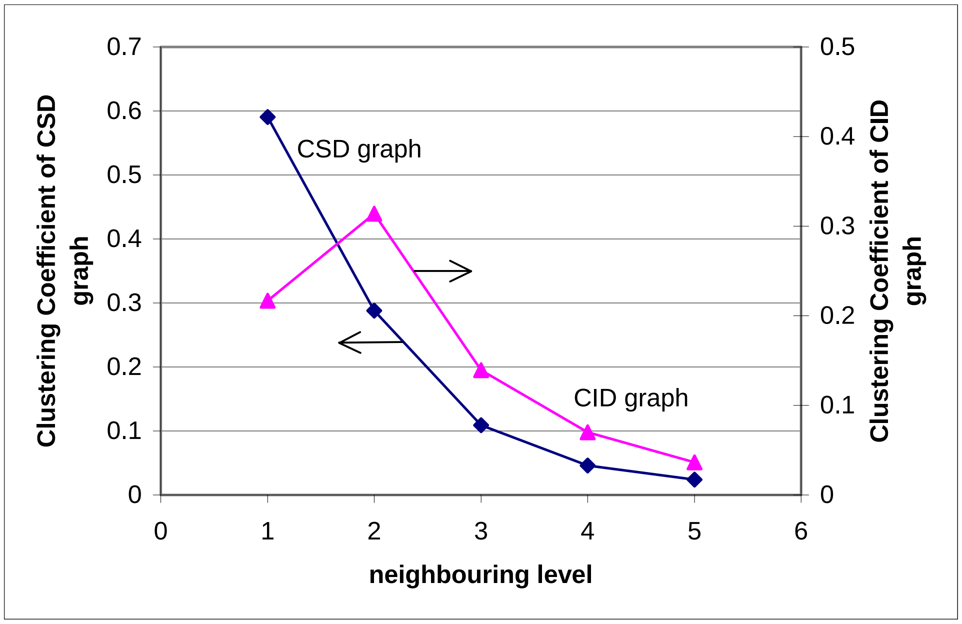

Figure 2. 8 Clustering coefficient at different level in CSD and CID graph.

Overall, we think both CID and CSD are some kinds of “small worlds” with very different structures. If we put CID and CSD between two extremes of random graph and regular graph, CID is much closer to a random graph while CSD inclines towards a regular graph.

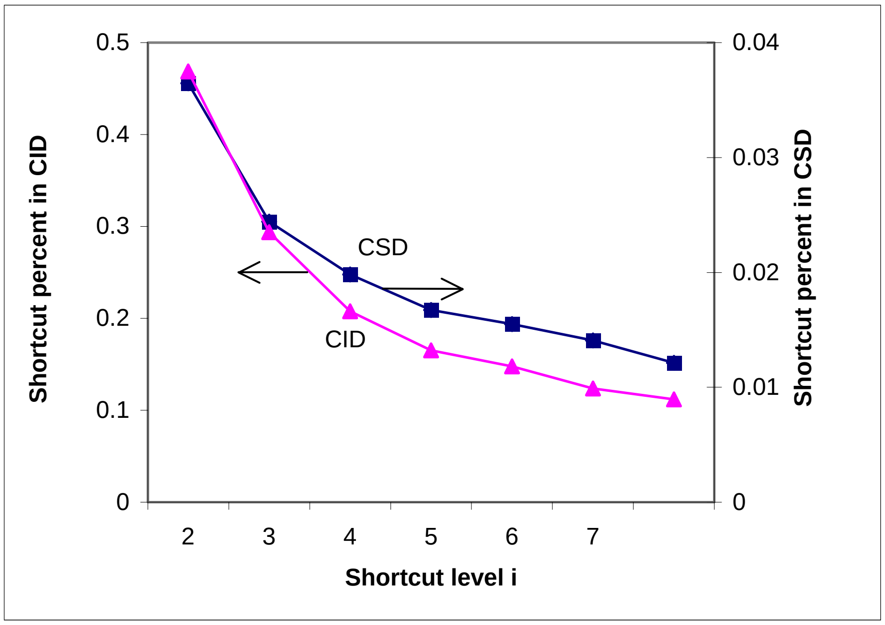

Figure 2. 9 Shortcut percentage at different level in CSD and CID graph.

Chapter 3

Clustering, Graph Coarsening and Drawing

Traditionally, proteins are studied and characterized individually based on their associated interactions, biochemical functions, genetic properties etc. However, recent studies show that a global view of the proteins interaction network may reveal more unidentified information such as the relationship between lethality of a protein and its centrality in the protein interaction networks[16]. A graphical layout for such network would be very helpful to get hidden information from it intuitively. Therefore, one objective in this thesis is to display the overall structure of our CID/CSD graph and obtain a global view of this network.

In graph visualization, one key issue is the size of the graph to be viewed. The number of nodes in the graph to be displayed is limited not only by the viewing platform, but also by the discernability of the viewer. In general, displaying an entire large graph might bring the viewer some sense of the overall structure of the graph or a node's environment within the graph, but makes it difficult to discern the details. According to a recent graph drawing competition, the number of nodes in a classical graph that can be best displayed on current visualization systems is limited to a few hundreds[17]. However, information visualization applications in the biological world such as our CID/CSD graph are typically associated with thousands or even millions of nodes. In order to reduce the visualization complexity, many "abstraction" and "reduction" methods have been proposed by researchers. One common approach is through clustering.

3.1 Objective of the Graph Coarsening

Clustering is also called *cluster analysis, grouping or classification*. Basically, clustering is the process of discovering the groups or classes among many elements based on certain criteria (or *semantics* in some literature). It has application in biology, economics, psychology and many other fields. In the context of graph visualization, clustering can be utilized to group related nodes into "*super nodes*" so that the overall visualization complexity decreases by restricting our view to those "*super nodes*". The whole procedure can be looked as the coarsening process to build a smaller and simpler layout version from a more complex and larger graph while maintaining the "backbone" structure.

However, "backbone" is only vaguely defined. To evaluate the performance of clustering, there exist many criteria such as minimizing the cross edges between members in different clusters, keeping the total weight in each cluster nearly same. In this thesis, our objective is to view a simpler "backbone" structure of the graph while keeping the local information inside each cluster available. Thus, the goal of our clustering algorithms is to group nodes in CID/CSD graph into disjoint subsets, called "*super nodse*", such that two criteria are satisfied:

1) Homogeneity: nodes in the same "*super node*" are closely related with each other, in other words, the clustering coefficient inside a "*super node*" is high;
2) Separation: nodes in different "*super nodes*" have little connection in the original CID/CSD graph.

In Chapter 2, the network properties of CID/CSD graphs have been studied. The results show that they both have very low characteristic path lengths comparable to that of random graphs with similar vertex size and average link numbers. However the clustering coefficients are much higher than what we would expect for a random graph. This implies that our CID/CSD graph is a “small-world”. In the case of CSD, with a low average edge number of 13 and total vertex number of 3725, the high clustering coefficient(~ 0.59) is ascribed mostly to the locally highly connected spot rather than the global clustering properties. To be more specific, inside this protein connection network, there are many densely connected clusters, linked together by shortcuts to achieve low characteristic path length. This is quite similar to our everyday social network. Most of us live in a specific social life cycle, and people inside this cycle are familiar (linked) with each other. However, most of the people don’t have too many connections with outside cycles. It is some “important” peoples’ friendships (connections) that bring the different cycles connected so that we can get the so-called “six degree separation”. These important persons (shortcuts) would be critical to the overall network structure and be part of the backbone of the network.

Based on the above observation, it would be feasible to utilize these shortcuts within the graph and discover the naturally occurring clusters since the percentage of shortcut is really small inside a “small world”, comparing to the cases that every edge might be shortcut inside a random graph and no shortcut exists in a regular graph. Therefore, we can further detail our graph coarsening problem as following.

Given a complex connected graph G(V, E), we want to build another connected graph G′(V′, E′) which G′(V′, E′) satisfies the following requirements:

1) Each $\nu \in V(G)$ has a corresponding ν' in $V'(G)$, *i.e.* $\nu \Leftrightarrow \nu'$, in the sense that ν' can be regarded as a subgraph of G and being represented as a "*super node*" in G′. This relationship is many-to-one.
2) For every edge $(\mu, \nu) \in E(G)$, if $\nu \Leftrightarrow \nu'$, $\mu \Leftrightarrow \mu'$ and $\mu \neq \nu$, then edge (μ', ν') should also exist in G′.
3) If two nodes of G belong to the same "super nodes" in G′, they should have a very close link. To be more specific, if $\nu \in V(G)$, $\omega \in V(G)$, $\nu \Leftrightarrow \nu'$, $\omega \Leftrightarrow \nu'$, $\nu' \in V'(G)$, and edge $(\nu, \omega) \in E(G)$, there should exist at least another path from ν to ω with length less than r through some nodes in the corresponding subgraph of ν' in the absence of edge(ν, ω) . r depends on how far we want to coarsen the graph. Larger r implies a simpler G′ but not guaranteed.
4) For every edge$(\mu', \nu') \in E'(G')$, if $\nu \Leftrightarrow \nu'$, $\mu \Leftrightarrow \mu'$ and $(\mu, \nu) \in E(G)$, then either edge (μ, ν) is a shortcut at level of r, or μ and/or ν are cut nodes in G.
5) The edge number inside the G′ should be minimized.

Requirements 1 and 2 will maintain the overall structure and connectedness properties of the original graph. Requirement 3) keeps the nodes inside a natural clusters more likely to be included in the same "*super node*". While the 2nd requirment makes a clear relationship between edges of the two graphs, the 4th one implies that all the edges in G′ exist because they are critical in connecting the original graph G such that their absence will make the distance between nodes belonging to different "*super node*" larger. Requirement 5 will reduce the display complexity of the resulting graph G′.

The general objective of our graph coarsening is to group “clustered” vertices together and reduce the number of vertices presented in graph visualization. In our algorithm, we utilized the distribution of shortcuts in G to identify those clusters. The clusters in G are defined as following:

> Let G be the original graph, and G1 be the graph obtained by removing all the shortcuts endpoints and cut nodes. Then each connected components in G1 is called a cluster in G.

Obviously, these clusters would satisfy our requirement 3) and denote “*super nodes*” in our coarsened graph. The “super nodes” combined with part of those shortcuts endpoints and cut nodes consistute our coarsened graph. Details can be found in the later section. Using this algorithm, coarsened graphs of CID/CSD have been built and the graph layouts are displayed using force-directed method. A java program has been implemented to provide the user with a graphical view of the CID/CSD graph at both local and global level.

The whole graph coarsening process can be divided into four phases:

1) Shortcut and cut node discovery;
2) Clustering;
3) Graph coarsening .
4) Graph drawing;

3.2 Clustering, Graph Coarsening and Drawing

3.2.1 Shortcut and Cut Node Discovery

Shortcut and cut nodes are defined in Chapter 2. The most important feature of them is that their absence will cause a longer path length among their neighbors. The problem to find out whether an edge(μ, ν) $\in$ E(G) is a shortcut(r) of G is similar to the question that if μ can be reach ν within r links without edge(μ, ν). (We only consider undirected and unweighted graph). This can be solved by a *depth-limited search* (DLS) with a little modification.

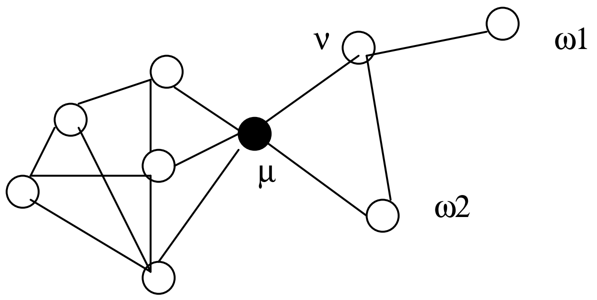

Figure 3. 1 μ is a cut node and connects the left and right group of nodes.

In the case of cut node analysis, those nodes with less than 4 links in G can be filtered out. The reason behind this strict "at least 4 links" gives credit to our objective of finding those critical nodes that either connects members between 2 larger groups or contributes to the overall graph connection. As shown in Figure 3.1, obviously, node μ plays a more important role than ν to bridge the left and right side. Similarly, cut nodes could be found by modified DLS with preprocess of finding neighbors. The pseudo codes of our algorithms are presented as following:

Algorithms:

//Function isShortcut(u,v,r) will decide if edge(u,v) is a shortcut at level r.

```
  isShortcut(u, v, r) {
        if ( u or v is leaf node or minor node)

              return false;

        push u into stack S;

        found = false;

        visit[u] = true;

        depth[u] = 0;

        if ( ! found && S is not empty){

           w = S.pop();

          if ( w = v ) found = true;

          if ( depth[w] < (r-1)){

              for each vertex z that adjacent to w{

              //modification of DLS to avoid edge (u,v) on the path

                 if ( !visit[z] &&  (z != v || (z = = v && depth[w] !=0) ) ){

                     S.push(z);

                     depth[z] = depth[w]+1;

        }}}}

        return found;

}
```

//Function isCutNode(v,r) will decide if node v is a shortcut at level r.

```
   isCutNode(v, r) {

        let L be the list of nodes adjacent with node v;
        if (size of L is less than 4) return false;
        count = 0;
        test = false;
        for all the pairs (u,w) in list L {
                if ( isReachable(u,w,v,r)) count++;
                if (count > = 2) return true;}
    return test;
}
```

//Function isReachable(u,w,v,r) will decide if node u can reach w within r links without using any links incident on v..

```
    isReachable(u, w, v, r) {
        push u into stack S;
        Found = false;
        visit[u] = true;
        depth[u] = 0;
        if ( ! Found && S is not empty){
            w1 = S.pop();
          if ( w1 = w ) Found = true;
          if ( depth[w1] < (r-1)){
                for each vertex z that adjacent to w1{
                //modification of DLS to avoid node v on the path
                   if ( !visit[z] &&  z != v ){
                    S.push(z);

                    depth[z] = depth[w]+1;

            }}}}

        return Found;

    }
```

Analysis:

In the worst case, the time complexity of DLS is *$O(b^l)$* and space complexity is *$O(b*l)$* where *b* is the branching factor of the search tree and *l* is the depth limit. However, branching factor *b* is not uniform for the search tree. On the average, degree of links *k* can approximately estimate the value of *b*, and the depth limit *l* equals to the shortcut level *r*. Thus, the time complexity of to find all the shortcut in graph G equals to *$O(|E|*k^r)$*.

To decide if a node ν is cut node or not, we perform DSL on each possible node pair combination from ν's neighborhood. Under the worst situation, this DSL has to be performed N*(N-1)/2 times, where N is the size of ν's adjacent list and can be estimated from degree of links *k*. Therefore, the total complexity should be expected as *$O(|V|*k^{(r+2)})$*. But the real case problem should have a much lower complexity that benefits from the preprocessing of "larger than 4 links" and the stop condition at "count = 2".

Overall the first phase complexity is *$O(|E|* k^r + |V|*k^{(r+2)}).) = O((|E|+ |V|*k^2)* k^r)$*.

3.2.2 Clustering

Algorithms:

The nodes involved in the shortcuts and cut nodes can be regarded as the “seeds” in our clustering process. First, each “future” cluster (not generated yet) is associated with an index *sid* and two list called *member[sid*] and *group[sid*], where *member[sid*] stores all the “seeds” presented in this cluster, and *group* keeps track of all nodes put into this cluster. The whole graph information is stored in array *member[]* and *group[]*, with *sid* as the index to this array. The largest *sid* is the cluster number in our graph. A non-seed node can only be associated with one *group* , while “seed” can be present in several *group*. The information of *sid* that each “seed” node is belongs to is kept in a 3rd list called *SEED[id]*, where *id* is the index which refers to a specific “seed” in G. List *SEED[id]* will be used during our later t connection stage.

To group related node into a cluster, each non-seed node traverse the graph by *Breadth First Search* (BFS) and stop further branching at each “seed”, i.e. not pushing it’s adjacent nodes in the queue. One exception is that its adjacent node is either leaf node or minor node.

For a node ν, if all of its adjacent nodes are “seed” nodes, then ν will not be grouped into any clusters by the above clustering process. Thus, special consideration should be given to this case. The following pseudo code gives detail information:

Analysis:

Using the above algorithms, the resulting clusters has the following properties:

1) For vertex $\mu \in V(G)$, $\nu \in V(G)$, if μ, ν belongs to different clusters and μ, ν is not the *seed node*, then there doesn’t exist such an edge (μ, ν) in G, i.e. $(\mu, \nu) \notin E(G)$.

In another word, all the links related to non-seed members is limited in this cluster, which will largely decrease the cross links between “super node” in the coarsened graph.

2) For vertex $\mu \in V(G)$, $\nu \in V(G)$, if μ, ν belongs to same clusters, edge $(\mu, \nu) \in E(G)$ and μ, ν is not minor node , a leaf node or “seed ” node, then there must exist an alternative path from μ to ν with length < r without edge (μ, ν) and must pass through at least one other member than μ, ν in the same clusters

```
Clustering (Graph G) {    //the edges in G is represented by a adjacent list
    for each vertex v ∈ V(G){
      identified[v] = false;
      sid = 0;                    //sid is the vertex representation of the cluster
      for(each vertex v ∈ V(G){
        if ( (  ! identified[v] && v is not a “seed” node ){
          put v into Queue Q, Q.Enqueue(v);
          visit[v] = true;
          while ( Q is not empty ){
              vertex w = Q.Dequeue();
              if( ! w is not a “seed” node ){
                  put w into member[sid];
                  identified[w] = true;
                  for (each vertex u which is adjacent to w in G )  {
                      if (!visit[u] ){
```

```
                    Q.Enqueue(u);
                    visit[u] = true;
              } }}        //end if
              else{
                put w into List group[sid];
                put sid into SEED[w];
                for (each vertex u which is adajacent to w in G ) {
                  if (!visit[u] && u is a leaf node ){
                    Q.Enqueue(u);   visit[u] = true;
               } } }                    //end else
           }}                     //end if
           sid ++;    }
    //process "seed" node without any "non seed" neighbors.
       for each node u in graph G{
           if ( ! idenfied [u] ){
              put sid into SEED[u];
              put nodes u into member[sid] and set identified[u] = true;
              sid++;}
       }
}
```

According to our cluster definition, this process is essentially equivalent to identifying all those connected components inside G1, which is derived from the original graph with all shortcut endpoints and cut nodes deleted. Each non "seed" node in the cluster will be traversed exactly once, while "seed" node might be visited several times which depends on the number of clusters it associated with. However, since no edge would be traversed more than once in this algorithms, this time complexity is much less than O(|E|).

3.2.3 Shortcut Connection

Algorithms:

Once we group related nodes into clusters, the next step is how to connect the otherwise disconnected graph built with shortcuts only and satisfy the 2nd requirement in our objective. In our algorithms, we introduce "super node" and "super edge". Basically, we merge all the non"seed" nodes inside a cluster into a "super node", and link two clusters based on their associated "seed" nodes. For those "seed" nodes associated with more than 1 clusters, i.e, size of SEED[id] > 2, it would be favorable to introduce a new single "super node" instead of make many connection between involved clusters, as illustrated in Figure 3.2. We call those links introduced owing to this reason as *pseudo shortcut*, shown as dashed link in Figure 3.2 .

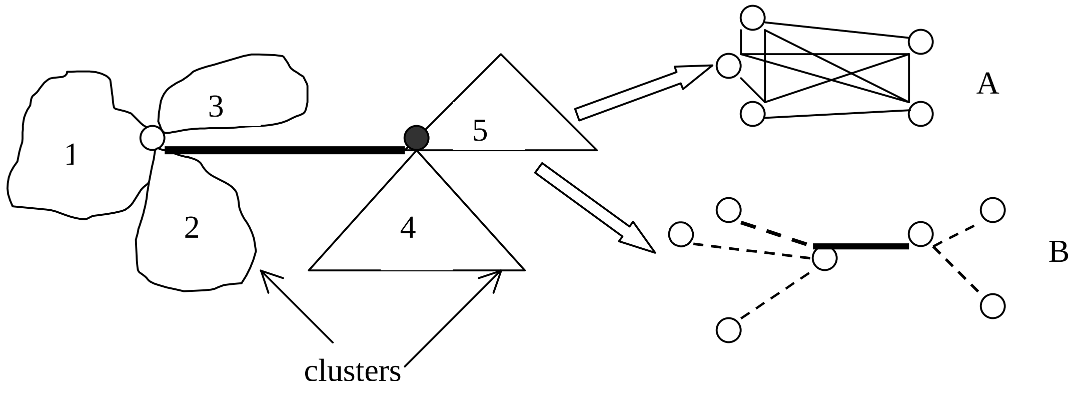

Figure 3. 2 Left side is the clusters connected through two "seed" node (black and white dot) associated with 3, and 2 respectively. Method A doesn't introduce any addition node and ends with 5 vertics and 10 edges. Method B add two "imaginary" nodes and results in 7 vertices and 6 edges. Overall, the layout of B is simpler than A.

The following is the detailed description:

// Connection() function will return a coarsened graph G′

```
Connection(){
    for each "seed" node u in G{
        if (size of SEED[u] > 2){
            for each element x in SEED[u]{
                //introduce pseudo shortcut;
                add edge(x, sid) into G′ ; }
                empty SEED[u] and then add sid into SEED[u];
                put u into member[sid] and group[sid];}
                sid++;
    }}
    for (each shortcut (u,v) in G ){
        for each member x in SEED[u]{
            for each member y in SEED[v]{
                    add edge(x,y) into G′  if not exist;
    }}}
}
```

Analysis:

By now, the skeleton of our coarsened graph has been achieved. Compared to the original graph G, this coarsened graph consists of vertices from two categories: 1) "*super node*" representing a cluster, i.e., individual connected component inside G1 (G1 is derived from G with all shortcut endpoints and cut nodes deleted). 2) Cut nodes and part of shortcut endpoints. Let G2 be the graph derived from original graph G with all shortcuts deleted and shortcut endpoints preserved. We define n as the number of clusters that can be reached from shortcut endpoint μ in G2 with a single link. If n = 0 or n >1, then μ would be shown in coarsened graph. If n =1, then μ will be hidden in the associated "*super node*". Obviously, all the edges in the coarsened graph correspond to either the shortcuts in G or *pseudo-shortcuts* introduced in our algorithms.

The major task of this Connection() is to go through all edges in G and find the corresponding links in G′ based on our criteria of requirment 2. Therefore, time complexity is O(|E|).

3.2.4 Graph Drawing:

Given a graph with a set of nodes and edges, graph drawing problem deals with the calculation of position of the nodes and the curve to be drawn from each edge so that a pleasing drawing of the graph could be provided. Among various graph drawing techniques, *Force-Directed Method*, also called *Spring Layout* algorithms, has been used successfully to produce well-balanced layout for undirected connected graph. In this algorithms, nodes of a graph are modeled as charged particles which impose repulsive forces on one another, and the edges of the graph are viewed as springs causing attractive forces between adjacent nodes of the graph. The algorithm was first proposed by Eades[18]. After that, many different physical models has been introduced to improve this algorithms. In this thesis, we design our own graph drawing code based on Kamada and Kawai's model(KK)[19]. Each node in KK's model is viewed as particles connected by springs whose ideal lengths $l_{i,j}$ equals to the graph-theoretic distances $d(i,j)$ between their two end-point particles multiplied by the desirable length of one edge we want to display in our layout. The goal of this KK model is to find a spring system with balanced energy *E* where *E* is defined by

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} k_{ij} (r_{i\,j} - l_{i\,j})^2 / 2$$

where $k_{i,j}$ denotes the strength of the spring between these two vertices and $r_{i,j}$ denotes the real distance between vertices i, and j in the layout.

In the coarsened graph, there are basically two types of links: shortcut and *pseudo-shortcut*. In the original graph, clusters linked to same nodes through *pseudo-shortcut* should be very close to each other, i.e. they share at least one common "seed" node. Thus, we define the desirable length of a single shortcut 4 times of that of one *pseudo-shortcut* when computing *d(i,j)* to reflect this difference.

Based on the above algorithms, we implemented a java program which take a input graph like the CID or CSD in text format, compute and display a coarsening graph based on the above clustering, coarsening, and drawing methods. Some major function in this program includes:

1) Computing the coarsening graph G′ at different level r ;
2) Display G′;
3) Display the local connection insider each "*super node*"(cluster), i.e., the corresponding subgraph of G.
4) Display the member information of the "*super node* ", such as the corresponding protein name in our CID or CSD network.

3.3 Results and Discussion

3.3.1 Comparison between CID and CSD graph

Table 3. 1 Summary of parameters in coarsened graph of CSD and CID

Graph	Original graph	Shortcut	Cut node	Shortcut only	Shortcut & Cut Node
CSD	|V| = 3,725; |E| = 23,986	660	1021	|V| = 241 |E| = 361 cluster =133 largest cluster size = 3056	|V| = 1032 |E| = 3872 cluster = 420 largest cluster size =886
CID	|V| = 1,297 |E| = 1,862	436	11	|V| = 297 |E| = 482 cluster =159 largest cluster size =68	|V| = 338 |E| = 562 cluster =182 largest cluster size = 64

At the beginning of this study, the above algorithm was designed as to start coarsening based on those shortcuts only (using only shortcut endpoints as our “seed” node). It works fine with CID graph. However, when it comes to the CSD graph, 3,056 out of 3,725 vertices inside CSD will be grouped into a single cluster under level r = 4. The total number of clusters generated is 133 linked through 361 edges, among which 97 clusters’ sizes fall into the range of 2-10. The result implies CSD is a highly clustered graph. Further study of CSD shows that there exist a significant number of cut nodes, which are not involved in any shortcut connection. Obviously, ignoring the existence of these cut nodes cause the whole graph collapse into a jumbo cluster with tiny ones surrounding it. Therefore, we modified our algorithms by bring those ignored cut nodes into our "seed" node consideration. At this time, total 420 clusters has been identified and the largest "clump" collapses into smaller pieces even though the size is still as large as 886. We believe that this largest cluster is inherent inside this CSD network. Surprisingly, nearly 1/6 of the vertices inside CSD falls into the cut node category. Table 3.1 lists some basic results of this algorithm. In this table, "shortcut only" refers to the results obtained if we only consider those shortcut to set up "seed" node and "shortcut & cut node" denotes the result using algorithms presented above. This table implies that the consideration of cut node have little effect on the clustering of CID while inducing significant change to the results of CSD.

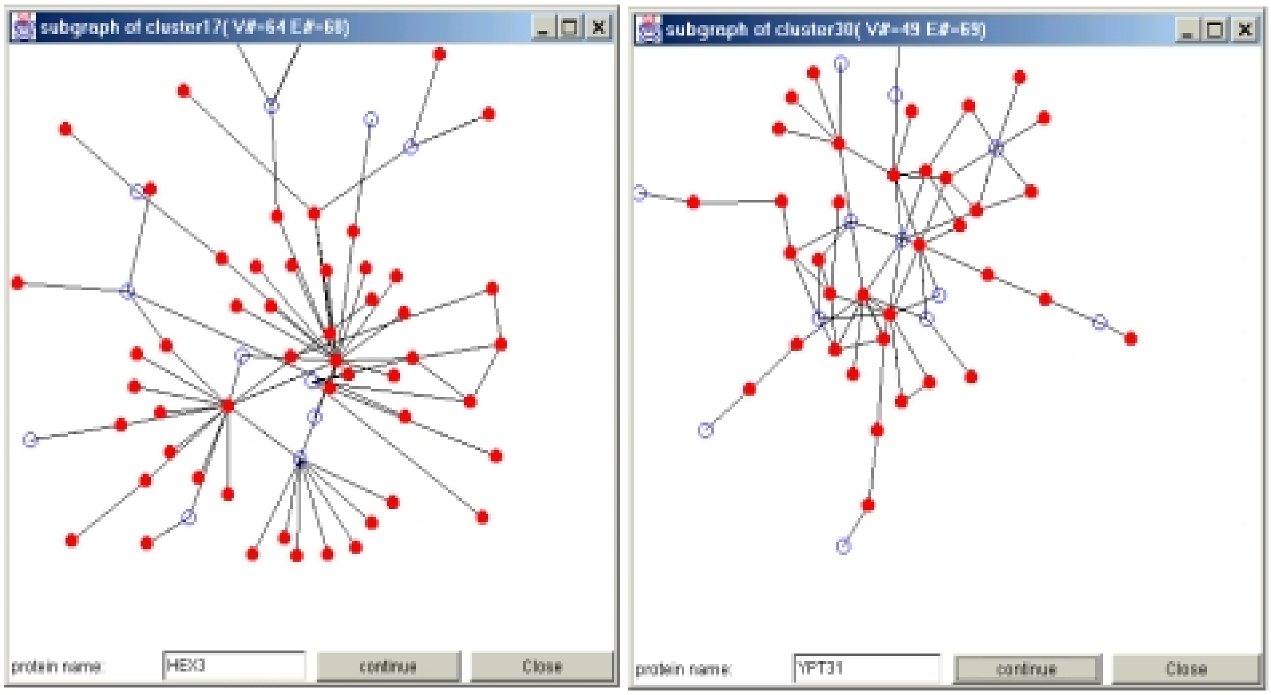
subgraph of cluster17(V#=64 E#=68)
protein name:
HEX3
continue
Close
subgraph of cluster30(V#=49 E#=69)
protein name:
YPT31
continue
Close

Figure 3. 3 Typical cluster structure of CID network. The red dots denote the non *seed* node while blue circles refer to the *seed* node. All the links of non *seed* node in original graph are presented. The *seed* node only shows those links inside this cluster. Each dot represent a protein, whose name can be displayed by double click on that dots.

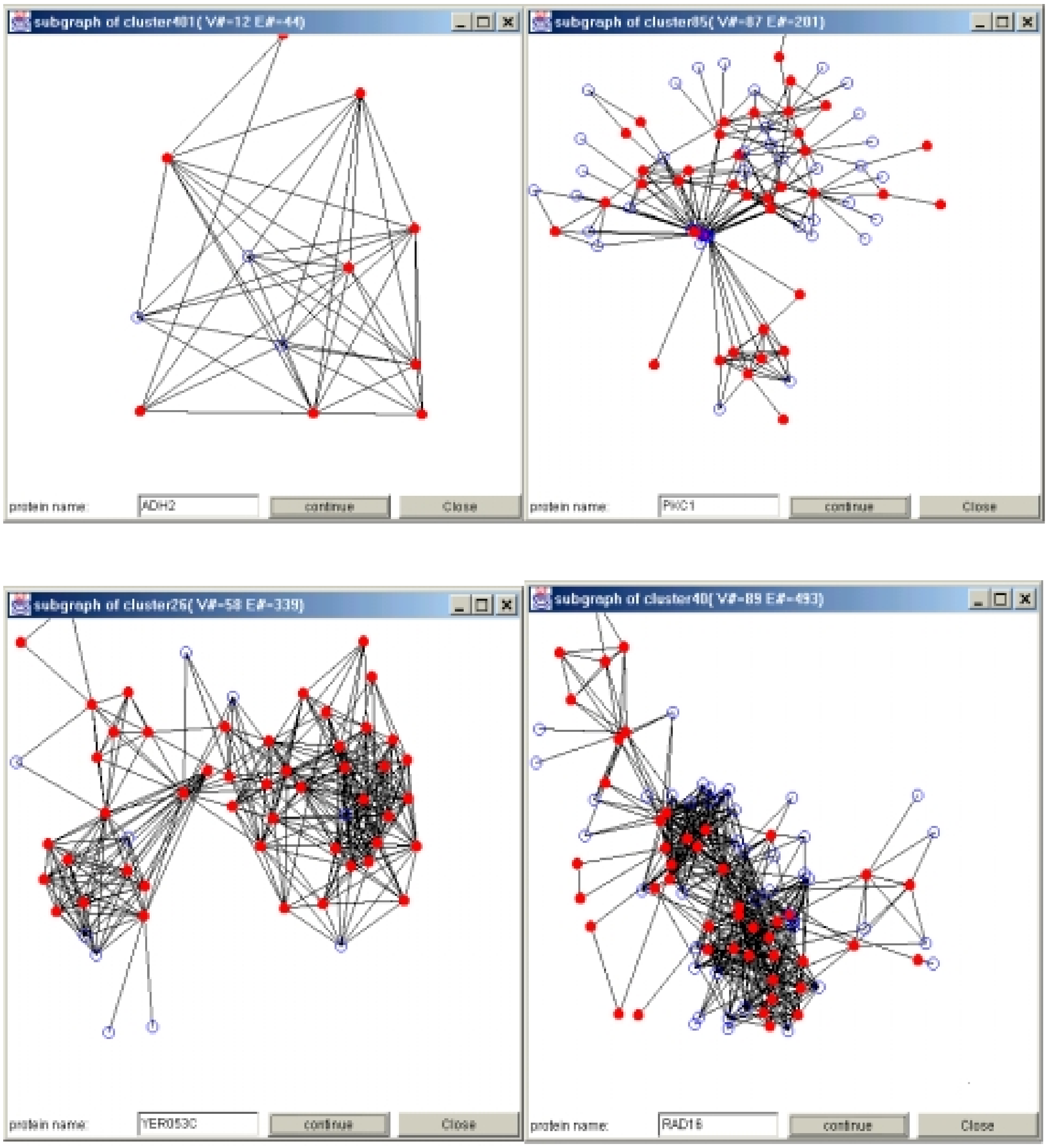
subgraph of cluster401(V#=12 E#=44)
protein name:
ADH2
continue
Close
subgraph of cluster85(V#=87 E#=201)
protein name:
PKC1
continue
Close
subgraph of cluster26(V#=58 E#=339)
protein name:
YER053C
continue
Close
subgraph of cluster40(V#=89 E#=493)
protein name:
RAD16
continue
Close

Figure 3. 4 Typical cluster structure of CSD network. The red dots denote the non *seed* node while blue dots refer to the *seed* node. All the links of non *seed* node in original graph are presented. The *seed* node only shows those links inside this cluster. Each dot represent a protein, whose name can be displayed by double click on that dots.

Figure 3.5 and Figure 3.6 illustrate the layout of the coarsened graph (at level r =4) of CID and CSD respectively. For CSD, the amount of edges is too large to be displayed even after coarsening, thus we hidden the edges to decrease the layout complexity. Some of their typical clustering structures are shown in Figure 3.3, and Figure 3.4 respectively.

Several things can be pointed out from these pictures. First, it is obvious that CSD network has a much more dense topological structure at both local area of a cluster if it exists, and the overall structure. That might be one of the reasons that account for a lower characteristic path length of CSD than CID even the former has more vertices involved. Secondly, the size distribution of most clusters is well balanced among the whole graph if we ignore that only "jumbo" cluster presented in CSD. Third, comparing to the layout of same network without any coarsening, as presented in Uetz 's paper[20], our coarsened CID network layout works relatively well to display the overall structure of a network with thousands of nodes.

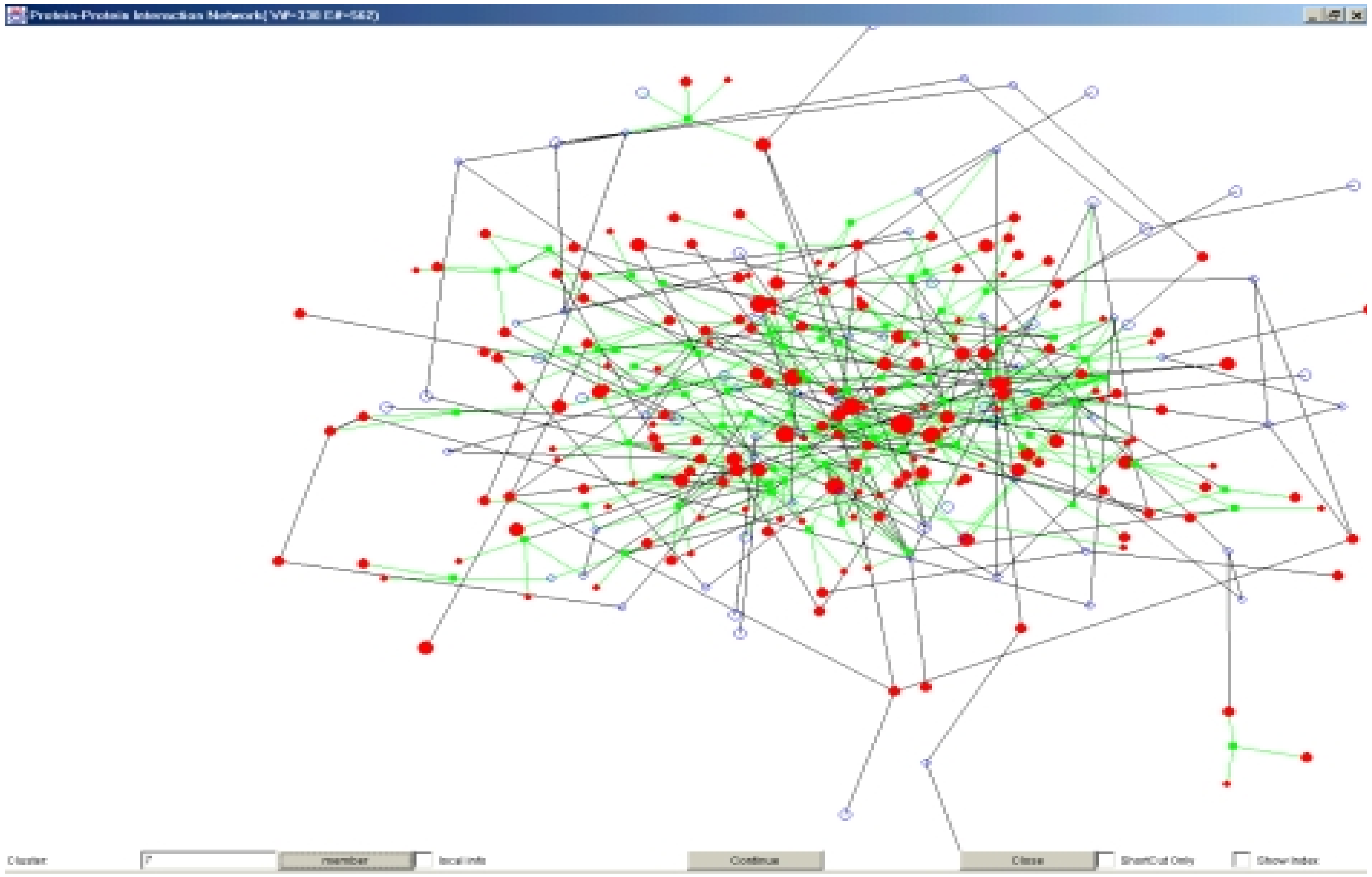

Figure 3. 5 The layout of coarsened graph of CID network. Red dots refer to the real clusters, blue circles denote the shortcut endpoints with n=0, and green diamonds represent those cut nodes and shortcut endpoints with n >1. The size of each dot denotes number of nodes associated with this cluster. The larger the size, the more nodes the cluster has. Green line (light color line) indicates that this link is “*pseudo-shortcut*”. Press button “ continue” may give a better view of the graph. Click on “shortcut only” will only display those shortcuts. Index of each cluster will be displayed in the Textfield beside label “Cluster” by double click those vertices. Press button “member” will display the subgraph corresponding to this cluster. Click on “local info” will only display this cluster’s links on the coarsened graph.

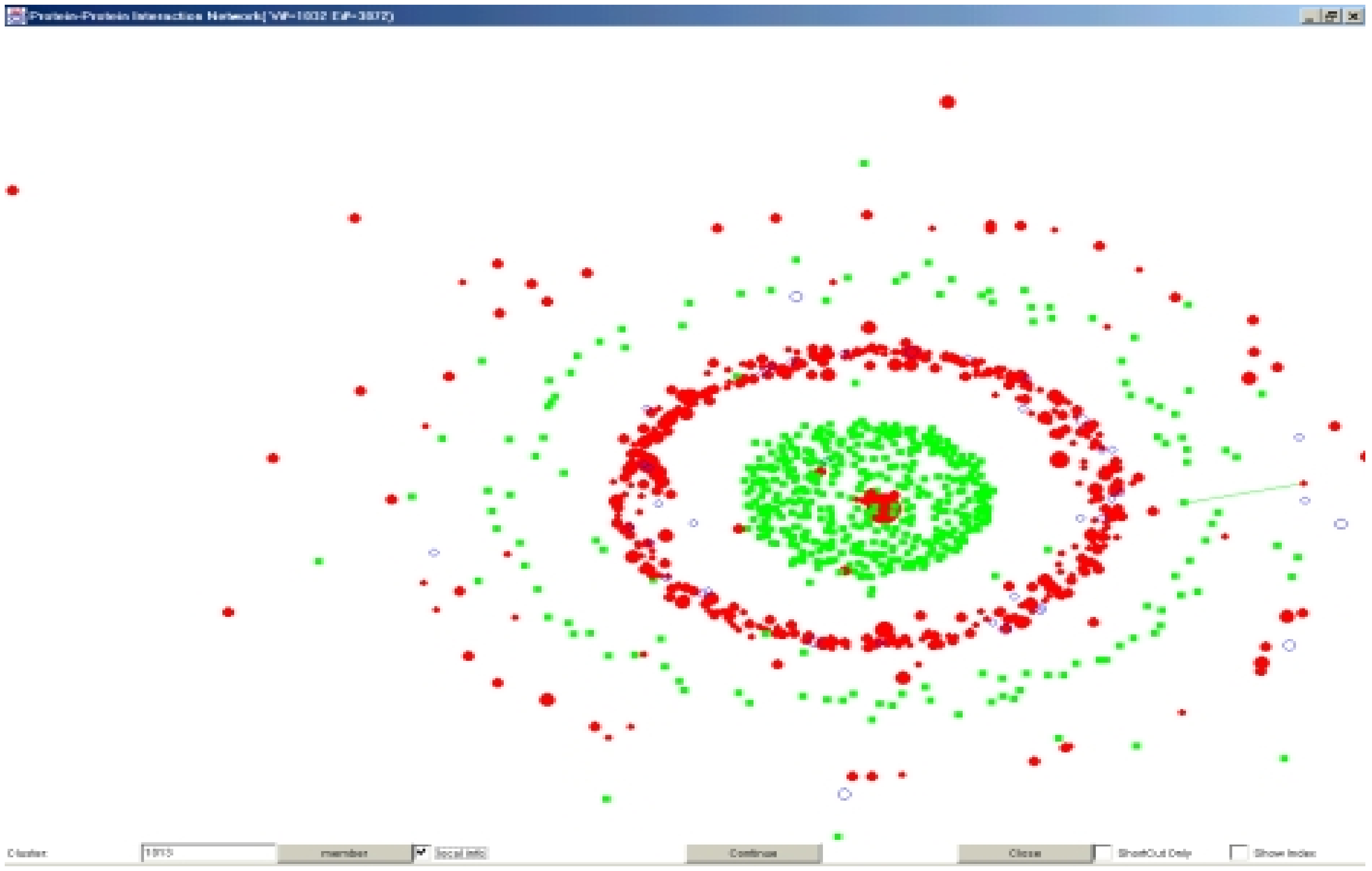

Figure 3. 6 The layout of coarsened graph of CSD network. Red dots refer to the real clusters, blue circles denote the shortcut endpoints with n=0, and green diamonds represent those cut nodes and shortcut endpoints with n >1. The size of each dot denotes number of nodes associated with this cluster. The larger the size, the more nodes the cluster has. Press button " continue" may give a better view of the graph. Click on "shortcut only" will only display those shortcuts. Index of each cluster will be displayed in the Textfield beside label "Cluster" by double click those dot. Press button "member" will display the subgraph corresponding to this cluster. Click on "local info" will only display this cluster's links on the coarsened graph. Edges are hidden since that amount is too large to be displayed.

3.3.2 *Biological Classification of Clusters*

According to Yeast Protein Database[21](YPD™), there are total 6,108 proteins discovered in yeast *Saccharomyces cerevisiae* by the time we collected data in January 2001. These yeast proteins have been assigned by YPD™ to different categories base on four criteria:

1) *Cellular Role* refers to the major biological process where the protein involves in, such as "protein folding", " Energy generation". Most of the published function prediction actually refers to this cellular role prediction. In this thesis, we'll stick to this tradition and "function" in our context mainly refers to cellular role if not specified.
2) *Molecular environment* consists of 9 members and denotes the molecular environment where a protein is located within the cell, e.g. "Soluable" and "RNA-associated".
3) *Subcellular localization* refers to the specific site in a cell where a protein is located and consists of 22 members such as " secretory vesicles" or "nuclear pore".
4) *Biochemical function* describes the principal structural, regulatory, or enzymatic function of the protein such as "ligand", "pheromnoe", or "DNA-binding protein" and consists of 57members.

The January 2001 version of YPD™ included 3649(59.74%), 2507(41.04%) , 3070(50.26%), 3234(52.85%) proteins with labeled "*cellular role*", " *molecular environment*", "*subcellular localization*", and "*biochemical function*" respectively. This YPD™ classification is broad and many proteins are assigned with more than one category in the same criteria. The detail information about this is summarized in Table 2. Among these 4 criteria, “cellular role” and “biological function” will be closely related to the protein-protein interactions. Therefore, we will mainly focus on these two in our following discussion.

Table 3. 2 YPD™ Categories Information.

Total 6108 proteins

Criteria	Member Number	Number of Labelled Proteins	Percentage of Labeled Proteins	Percentage of Labeled Proteins with more than 1 assignment
cellular role	43	3649	59.74%	41.08%
molecular environment	9	2507	41.04%	6.42%
subcellular localization	28	3070	50.26%	27.43%
biochemical function	57	3234	52.95%	47.59%

In our graph coarsening algorithms, several clusters with significant size have been identified. Spontaneously, our next question is that whether nodes (representing proteins) inside each cluster have some biological relationship or they just group together by chance. To test this, we study the biological features insider the chosen cluster under the overall network backgroud. Here is our general strategy:

1) Randomly choose three cluster containing more than 20 nodes as our study cases.
2) Calculate the protein functional distribution over the whole network based on certain criteria like “cellular role” etc. Here the “functional” refers to a specific type of a classification criteria. For example, the “cellular role” of protein “PDE1” is “aging”, with “hydrolase” as its biological function. To be more specific, we calculate how many percentage of proteins falls into “aging”, “protein folding”

and so on. Based on these data, we can obtain an overall distribution for the "cellular role". Same thing is done for "biological function".

3) Perform the similar computation as 2) at each cluster.

4) Let $P_{all,i}$ be the percentage of a certain type in overall graph distribution, and $P_{c,i}$ be the one of cluster, we define the abnormal index α as:

$$\alpha = \frac{(P_{c,i} - P_{all,i})}{P_{all,i}}$$

$\alpha = 0$ if $P_{c,i}$ is less than $P_{all,i}$ and i refers to a specific functional type. This α can be regard as an index of how the occurrence of this functional type is unique for this cluster.

5) For each cluster, we choose the top three types with highest counts of occurrence.

Besides our coarsening algorithms, the reliabilities of our data source will also have a great effect on the answer to this question of relationship between topological structure and biological interactions. Since CID network derived from data that has been widely accepted and verified, we will limit our basic biological exploration to CID network. The results are listed in Table 3.3.

Table 3.3 convincingly proved the previous assumption proposed by biologist[5] that protein with similar function (in broad sense as referring to both "cellular role" and "biological function") is more likely to be clustered together. It might also suggest some relationships between the most "popular" functional types inside the same clusters. However, the proof of this hypothesis is out of the scope of this thesis and beyond the author's limted biological knowledge.

Table 3. 3 List of top 3 "fuctional" types inside three clusters with different size and their abnormal indices

Criteria	Cluster index	Number of nodes in cluster	Functional name	Occurrence	α
cellular role	5	43	Carbohydrate metabolism	7	3.745715
			Cell cycle control	5	0.765519
			Pol II transcription	5	0.377966
	16	68	Mating response	24	4.694857
			Cell polarity	22	4.220286
			Signal transduction	18	4.089021
	55	20	Protein translocation	5	7.405871
			Cell stress	5	3.371053
			Protein folding	11	12.73759
Biological function	5	43	Transcription factor	7	1.296
			Complex assembly protein	6	6.028571
			Hydrolase	4	0.009231
	16	68	Protein kinase	10	3.270834
			Transferase	10	0.990291
			GTP-binding protein/GTPase	7	3.948276
	55	20	Chaperones	11	10.5641
			Heat shock protein	8	16.94188
			Hydrolase	5	0.293886

Chapter 4

Protein Category Prediction

The process of finding the function of a protein usually involves a lot of genetic and biological experiments if there is no protein available with similar amino acid sequence and known functions. However, as more data have been collected into protein databank, new strategies become possible.

Marcotte et al[22] categorized the proteins by correlated evolution, mRNA expression patterns, protein interaction, and patterns of domain fusion to generate of 93,750 functional links with different "confidence quality" among 4,701 proteins of the yeast *Saccharomyces cerevisiae*. Using these functional links, they provided general function assignment to more than half of the 2,557 previously uncharacterized yeast proteins. Among these, 374 (or 15%) were assigned a general function from the high and highest confidence functional links. After that, Fields group[23] at University of Washington performed a global analysis on 2,709 published interactions between proteins of the yeast *Saccharomyces cerevisiae* and established a single large network of 2,358 interactions (functional links) among 1,548 proteins, i.e., the **PID** network used in this thesis. Based on the known functions of the interacting partners, they correctly predicted a functional category for 72% of the 1,393 characterized proteins with at least one partner of known function. 394 previously uncharacterized proteins have been predicted.

Biological experiments show that protein coded by a similar sequence would be functionally linked. Based on this consideration, we introduced similar-sequence protein links into our version of protein connectivity network **SID**, in addition to protein-protein interaction data. It is expected that we can provide more accurate functional prediction to more unknown proteins using this **SID** map.

Of the total 6,108 yeast proteins, similar functional proteins cluster into a specific region of the large network. In another word, if two proteins have interaction or/and sequence similarity, they have the high probability to share the same function class. This has been partially proved in section 3.3.2. Table 4.1 shows the percentage of links in **SID** with two proteins that have been characterized and share at least one common function. It implies that 60.25 % of the links in **SID** graph with two proteins characterized can also be considered as "functional similarity" links. Further analyses also demonstrate that proteins that are found in the same location or in the same complex (molecular Environment) or with the same biochemical function are likely to be involved in the same or related cellular process(functions). For example, Table 4.1 shows that among 12478 proteins links that have both function and subcellular location characterized, 8365 have similar subcellular location and 76.12% of these 8365 links have same function.

Table 4.1 also lists such statistical data for CSD graph. Not surprisingly, the CSD displays very close statistical features to SID with regard to the "functional similarity comparison". Noting that our CSD graph is the largest component in the unconnected SID graph and contains only ~60% of the original SID nodes, we may say that this "functional similarity " applies to every region of the SID graph. If we compute the percentages of the links with similar function among different regions of the graph, they should be very close to each other. It follows naturally that such conclusion would also apply to those area with proteins uncharacterized. Therefore, we can use this protein network to predict, or at least, narrow down the possible functional assignment of an unknown protein based on its connectivity with other characterized proteins.

Table 4. 1 Function Similarity Analysis of SID graph.

	SID (total 6108)proteins		**CSD** (total 3725 proteins)	
Criteria	%	Characterized Links**	%	Characterized Links**
Cellular role	60.25%	17010	59.85%	16696
Cellular role& molecular environment	86.64%	9367	85.35%	9179
Cellular role & biochemical function	80.85%	13451	78.46%	13160
Cellular role & subcellular localization	76.12%	12478	74.17%	12257

**Characterized Links-----number of links consists of two proteins that have been characterized with respect to criteria.

4.1 Method I----Simple Function Comparison

To prove the correctness of this strategy and assess the reliability of this SID network, we determine how well this SID network can be used to predict the functions of those already characterized proteins following the similar method proposed by Fields group[1]. Here is the detail description: Let P be the protein under consideration, for each protein that has link with P in the **SID** graph, if it has been characterized, put its function into the candidate function list. Then, we count the how many times each function class appear in the candidate function list, and choose the first 3 with highest count (> 0) as our predicted functions for P. After that, we compare our prediction and the actual functions of P, if they share at least one, then we regard this prediction for P as correct. By this method, the **SID** allows a correct prediction for 82.67% of the 3116 characterized proteins with at least one characterized partner. This can be compared to 72% correctness of the 1,393 characterized proteins obtained by Fields group.

4.2 Method II----Cross Function Links

During our study, we noticed that cross-function links also exist in our **SID** database. For certain functions like “septation”, they even don’t have similar function links, i.e., a “septation” protein is more likely to have relationship with a different functional protein rather another “septation” protein. Let i, j denote protein function, e.g. “aging”, “meiosis”, we count the number of links for each combination of function among **SID** network and denotes this number as count(i,j). If $i \neq j$ and count(i,j) > count(i,i)/2, then we regard this cross-function link between i and j is significant(‘1’ shown in Table 4.2). As shown in Table 4.2, even though the similar function linkage dominated in this table (larger number along the diagonal inside the matrix), some cross-function linkages are also significant to be noticed. For example, if a protein has function “aging ”, its partners (the proteins that have links with P in **SID** network) are highly possible to possess function of same function of “cell cycle control” or “chromatin” function in addition to protein with “aging” function. To account for this, we modify our prediction Method I as following:

1) First, we count the number for each possible cross-function link count[i][j], including similar-function link, among SID graph ;

```
for each link(P1, P2) in SID{
    if ( both P1 and P2 are characterized){
        for each function i, j that P1, P2 possess respectively
```

```
                count[i][j]++;
        }}}
```

2) To predict the function of a protein P with at least one characterized partner, we put all the functions that P1, which has direct link with P in SID and has been characterized, into a candidate list. We repeat this for every qualified partner of P and end up with a long list. The total number of functions in the candidate list is denoted as NUM; Then, we compute the number of appearance for each possible function, function[i], in this list.

3) After that, we calculate the score for function i that protein P might possess as following:

$$\text{total[i]} = \sum_j count[i][j];$$

$$\text{Score[i]} = \sum_{\text{j}} (\text{function[j] - NUM * count[i][j]/total[i]})^2$$

4) Finally, we choose our prediction for protein P as the top 3 ones with lowest score.

Using this method, the **SID** makes a correct prediction for 79.07% of the 3116 characterized proteins with at least one characterized partner. This number is a little bit lower than our simple prediction method I. However, if we broaden our prediction range, i.e., we choose the top 10, instead of 3 with highest score or frequency, method II will make a correct prediction for 96.63% of the characterized proteins, comparing to 88.96% obtained by method I. This suggest that method II be good for narrowing down the prediction range or to exclude those false functions but not suitable to predict correct function in a short range.

Table 4. 2 The significant level for each combination of cellular role in SID network

		AA	Am	Cm	Ca	Cc	Cp	Cs1	Cs2	Cw	Cs	CC	Dr	Ds	DD	Eg	Lf	Mr	MM	Mf	Mt	MM	Nt	Nm	OO	Om	Pm	Pl	Pl	Pl	Pc	Pd	Pf	Pm	Ps	Pt	Rp	Rs	Rt	RR	SS	St	Sm	Vt
Aging	AA	1				1					1																		1													1		
Amino-acid metabolism	Am		1																																								1	
Carbohydrate metabolism	Cm			1																																							1	
Cell adhesion	Ca			1	1	1		1	1	1	1				1			1				1							1													1		
Cell cycle control	Cc					1																							1															
Cell polarity	Cp						1											1																										1
Cell stress	Cs1							1		1																			1													1		
Cell structure	Cs2						1		1																																			1
Cell wall maintenance	Cw									1																																		
Chromatin/chromosome structure	Cs					1					1																		1															
Cytokinesis	CC					1	1			1		1						1				1							1															1
DNA repair	Dr					1					1		1	1															1															
DNA synthesis	Ds					1					1		1	1																														
Differentiation	DD*			1		1	1	1		1	1	1			1			1	1				1						1			1		1								1		1
Energy generation	Eg															1																											1	
Lipid, fatty-acid and sterol metabolism	Lf																1																										1	
Mating response	Mr					1	1											1											1													1		
Meiosis	MM					1					1		1					1	1			1							1			1		1			1			1		1		
Membrane fusion	Mf																			1																								1
Mitochondrial transcription	Mt															1					1																							
Mitosis	MM					1																1																						
Nuclear-cytoplasmic transport	Nt																						1																					
Nucleotide metabolism	Nm																							1																				
Other*	OO		1	1		1		1	1		1		1	1	1	1	1	1	1			1	1		1				1	1		1		1	1	1	1	1	1			1		1
Other metabolism	Om		1																							1			1														1	
Phosphate metabolism	Pm			1		1																					1		1														1	
Pol I transcription	Pl					1		1			1																	1	1	1														
Pol II transcription	Pl																												1															
Pol III transcription	Pl					1					1																	1	1	1														
Protein complex assembly	Pc					1			1		1											1							1		1		1										1	
Protein degradation	Pd																															1												
Protein folding	Pf																																1											
Protein modification	Pm																															1		1										
Protein synthesis	Ps																																		1		1							
Protein translocation	Pt																																1			1								
RNA processing/modification	Rp																																				1							
RNA splicing	Rs																																				1	1						
RNA turnover	Rt					1					1																								1		1	1	1					
Recombination	RR					1					1		1	1					1										1			1								1				
Septation**	SS																																											
Signal transduction	St																	1																								1		
Small molecule transport	Sm																																										1	
Vesicular transport	Vt																																											1

*fuction "Differenctiation" and "Other" has more cross-function links than similar function links
** Septation has no similar function interaction.

4.3 Method III----Prediction With Auxiliary Criteria

As mentioned above, proteins that are found in the same location or in the same complex(molecular Environment) or with the same biochemical function are more likely to be involved in the same or related cellular process. Thus, we expect to obtain a better correct prediction if we combine other criteria to build the candidate list. Based on this, we proposed another prediction method as described following:

Let P be the protein under consideration and C as our auxiliary criteria, for each protein P2 that has link with P in the **SID** graph, if it has been characterized for both function and criteria C, and if it share at least one feature with P with regard to criteria C, put its function into the candidate function list. Then, we count the how many times each function class appear in the candidate function list, and choose the first 3 with highest count (> 0) as our predicted functions for P. Table 4.3 compares all of these three methods. It shows that method III correctly predict 91.81% of 1930 characterized proteins by using molecular environment as auxiliary criteria, 88.75% of 2419 characterized proteins by using biochemical functions, and 88.14% of 2252 characterized proteins by using subcellular locations. All these three auxiliary criteria increase our correct prediction rate at the price of decreasing the number of proteins it can predict.

4.4 Summary

Table 4.3 also lists the correct prediction percentage and number of uncharacterized protein we can predict based on the CSD graph. The results are quite close to those obtained from SID graph. The approximately 1/5 false prediction from method I or less from others are likely due to the false links we included in our network, incomplete function annotation for a given characterized protein, or undiscovered connections so far etc. Even this minor false prediction might require us be cautious when making any interpretation of new inferences from this SID or CSD networks, the overall validity of network SID or CSD to predict function seems confirmed.

Table 4. 3 Summary of results from three protein prediction methods

		Correct Prediction %	#of characterized protein	# of unknown protein predicted
SID(6,108 proteins)				
Method I		82.67%	3116	907
Method II		79.07%	3116	907
Method III	Biochemical function	88.76%	2419	138
	Molecular Environment	91.81%	1930	65
	Subcellular Location	88.14%	2252	89
CSD(3,725 proteins)				
Method I		80.83%	2732	795
Method II		79.57%	2732	795
Method III	Biochemical function	88.52%	2090	126
	Molecular Environment	91.40%	1790	60
	Subcellular Location	87.17%	2027	81

Chapter 5

Conclusions

In this thesis, we built up two protein-protein relationship networks detailed at different level from real world data, i.e. PID and SID graph. To the best of our knowledge, SID network are the first one that combines the interaction and sequence similarity data. Based on the our study toward the global structure of them, the following conclusions can be made:

1) Topological studies of the largest connected component inside PID and SID network indicate that they both exhibit "small world" properties of clustering and small characteristic path length at different clustering level. If we put PID and SID between two extremes of random graph and regular graph, PID is much closer to random graph while SID inclines towards regular graph. In addition, power law relationship is also observed in both networks.
2) A graph clustering and coarsening algorithm based on the unique existence of shortcuts and cut nodes inside a "small world" has been proposed. The overall time complexity of this algorithm is $O(|E|+ |V|*k^2)* k^r)$, mainly due to the discovery of shortcuts and cut nodes. Layouts of the coarsened graph indicate that this algorithm works better for "small world" graph whose structure is characterized with "locally dense" and "globally sparse". In our study, it simplifies the display complexity of PID graph.

3) Topological clusters identified by our proposed algorithms demonstrated a highly degree of “functional clustering”. It is expected that future topological study of the network structure may also reveal some hidden biological connection.
4) Based on SID network, we developed three prediction methods for protein functional assignments. All of these three methods using our SID network provide higher prediction accuracy than the one reported in recent published paper.

Appendix

Source Code

The following is the major part of source codes of our graph clustering, coarsening and drawing algorithms.

Coarsen.java

```
/*coarsen()--read a rough connected graph with name of the node and interface with graph drawing
Author: Li Zhong
*/

import java.io.*;
import java.util.*;
import java.awt.*;

public class coarsen{
        static int MAX = 4000;
        static int MAX_I = 6500;
        static int MAX_G = 1500;
        int threshold1 = 0;         //the threshold between normal cluster and cluster with all shortcut;
        int threshold2 = 0;         //the threshold between cluster with all shortcut and abstract node;
        List edge[];                //the adjacent list
        List group[];               //the shortcut node with group i is connected with
        List member[];              //the node in the group i
        String name[];              //the protein name of node x
        //List group[];             //the groups which shortcut node is associated with

        boolean isFakeCut[];        //count for those vertex with less than 2 links.
        boolean scut[];             //decide if a vertex is shortcut vertex;
        int v_nbr = MAX;            //the total vertex number in the graph
        int e_nbr = MAX;            //the total edge number in the graph

        G g = new G();              //keep track of the short cut node
        G CG = new G();             //clustered graph

        int GroupNbr = 0;
        int cv_VIP = 0;

        coarsen(){
                edge = new List[MAX];
                group = new List[MAX];
                scut = new boolean[MAX];
                isFakeCut = new boolean[MAX];
        member = new List[MAX];
        name = new String[MAX];

        for(int i = 0; i < MAX; i++){
                edge[i] = new List();
                group[i] = new List();
                member[i] = new List();
                scut[i] = false;
                isFakeCut[i] = false;
                name[i] = " ";
                //member[i] = 0;
        }
}

void readFile(String filename){
        Runtime rt = Runtime.getRuntime();
        rt.gc();
        int index[] = new int[MAX_I];
        for ( int i = 0; i < MAX_I; i++)
                index[i] = -1;
        try{
                FileInputStream fis1 = new FileInputStream(filename);
                BufferedReader is1 = new BufferedReader(new FileReader(fis1.getFD()));
                String in1 = is1.readLine();

                int count = 0;
                while ( in1 != null){
                        if(in1.indexOf(".") != -1){
                        int from =  Integer.parseInt in1.substring(0, in1.indexOf(".")).trim());
                        index[from] = count;
                        count++;
                        }                                       //end if

                        in1 = is1.readLine();
                }                                               //end while
                is1.close();
        }                                                       //end try-catch
        catch (IOException e)  {
                System.out.println("File "+filename+" Not Found");
        }

        try{
                FileInputStream fis1 = new FileInputStream(filename);
                BufferedReader is1 = new BufferedReader(new FileReader(fis1.getFD()));
                String in1 = is1.readLine();

                int count = 0;
                int edgecount = 0;
                while ( in1 != null){
                        int from, to, from_i = -1, to_i = -1;
                        String temp, protein;

                        if(in1.indexOf(".") != -1 && in1.indexOf("->") != -1){
                        from = Integer.parseInt(in1.substring(0, in1.indexOf(".")).trim());
                        from_i = index[from];
                        protein = in1.substring(in1.indexOf(".")+1,  in1.indexOf(":")).trim();
                        name[from_i] = protein;

                        temp = in1.substring(in1.indexOf(":")+1).trim();
                while (temp != null && temp.indexOf("->") != -1 ){
                        to = Integer.parseInt(temp.substring(0, temp.indexOf("->")).trim());
                        to_i = index[to];
                        edge[from_i].insert(new Integer(to_i));
                        temp = temp.substring(temp.indexOf("->")+2).trim();
                        edgecount++;
                }                                                       //end while
                                count++;
        }                                                               //end if
                        in1 = is1.readLine();
}                                                                       //end while
is1.close();

v_nbr = count;
e_nbr = edgecount/2;
rt.gc();
}                                                                               //end try-catch
catch (IOException e)  {
System.out.println("File "+filename+" Not Found");
}  }

public List getChildAtDepth(int i, int limit){
        List L = new List();
        boolean visit[] = new boolean[MAX];
        int depth[] = new int[MAX];

        for(int j= 0; j< MAX; j++) {
                visit[j] = false;
                depth[j] = 80;
        }

        Stack S = new Stack();
        S.Push(new Integer(i));
        visit[i] = true;
        depth[i] = 0;

        while (S.getListNbr() != 0) {
                int cur = ((Integer)(S.Pop())).intValue();
                if(cur != i) L.insert(new Integer(cur));
                if ( depth[cur] < limit){
                        for(Cell c = edge[cur].getHead(); c != null; c = c.rightOf()){
                                int adj = ( (Integer)(c.objectOf()) ).intValue();
                                if(!visit[adj])  {
                                        S.Push(new Integer(adj));
                                        visit[adj] = true;
                                        depth[adj] = depth[cur]+1;
                                }
                        }
                }
        }
        return L;
}
```

```
//this function will find if node j is reachable from j with distance of limit at the
//absence of node exclude;
public boolean isReachable(int i, int j, int exclude, int limit){
        boolean visit[] = new boolean[MAX];
        int depth[] = new int[MAX];

        for(int k= 0; k< MAX; k++) {
                visit[k] = false;
                depth[k] = 80;
        }

        Stack S = new Stack();
        S.Push(new Integer(i));
        visit[i] = true;
        depth[i] = 0;

        while (S.getListNbr() != 0) {
                int cur = ((Integer)(S.Pop())).intValue();
                if ( cur == j ) return true;
                else if ( depth[cur] < limit){
                        for(Cell c = edge[cur].getHead(); c != null; c = c.rightOf()){
                                int adj = ( (Integer)(c.objectOf()) ).intValue();
                                if(!visit[adj] && adj != exclude && !scut[adj])  {
                                        S.Push(new Integer(adj));
                                        visit[adj] = true;
                                        depth[adj] = depth[cur]+1;
                                }
                        }
                }
        }
        return false;
}

public void cut(){
        boolean temp[] = new boolean[v_nbr];
        for ( int j = 0; j < v_nbr; j++)
                temp[j] =false;
        int cutnode = 0;
        for ( int i = v_nbr -1; i >= 0; i--){
                if ( !scut[i] && edge[i].getListNbr() > 4){
                        List L = new List();
                        for ( Cell c = edge[i].getHead(); c != null; c = c.rightOf()){

                                int adj = ((Integer)(c.objectOf())).intValue();
                                if( ! scut[adj] && edge[adj].getListNbr() > 2)
                                        L.insert(new Integer(adj));
                        }
                        if ( L.getListNbr() >= 4 ){
                        int count = 0;
                        boolean test = false;
                        for ( Cell c1 = L.getHead(); c1 != null && !test; c1 = c1.rightOf()){
                        int from = ((Integer)(c1.objectOf())).intValue();
                        for (Cell c2 = c1.rightOf(); c2 != null && !test; c2 = c2.rightOf()){
                        int to = ((Integer)(c2.objectOf())).intValue();
                        if (!isReachable(from, to, i, 4) )       count++;
                        if ( count >= 2) test =true;
                        }}
                        if ( test ){
                                scut[i] = true;
                                        g.insertV(i);
                                        cutnode++;
                                }
                        }
                }
        }                               //end for loop
        System.out.println("total cut node = " + cutnode );
}

//find all the 2-level child of i except path through j
public List getChild(int i, int exclude){
        List L = new List();

        boolean visit1[] = new boolean[MAX];

        for(int j= 0; j< MAX; j++)
                visit1[j] = false;

        for( Cell cur =edge[i].getHead(); cur != null; cur = cur.rightOf()){
                int to =(  (Integer)(cur.objectOf())  ).intValue();
                if(! visit1[to] && to != i && to != exclude){
                        L.insert(new Integer(to));
                        visit1[to] = true;

                for( Cell child = edge[to].getHead(); child!= null; child = child.rightOf()){
                int to_child = ( (Integer)(child.objectOf())  ).intValue();
                if(! visit1[to_child] && to_child != i && to_child != exclude) {
                        L.insert(new Integer(to_child));
                        visit1[to_child] = true;
                }       }                       //end for
                }                               //end if                }
        //end for
        return L;
}

public String getName(int i ){
        return name[i];
}


//test if two list have common field
public boolean isCross(List L1, List L2){

        boolean found = false;
        if(L1.getHead() != null && L2.getHead() != null){

                for( Cell c1 =L1.getHead(); c1 != null && !found; c1 = c1.rightOf()){
                        int i1 =(  (Integer)(c1.objectOf())  ).intValue();

                        for( Cell c2 = L2.getHead(); c2!= null && !found; c2 = c2.rightOf()){
                                int i2 = ( (Integer)(c2.objectOf())  ).intValue();
                                if(i1 == i2) {
                                        found = true;
```

```
                    }
                }
            }                                   //end for
        }                                       //end if

        return found;
    }

    public int getVN(){
        return v_nbr;
    }

    public int ShortCut(){
        Runtime rt = Runtime.getRuntime();
        rt.gc();

        int count = 0;
        int v_count = 0;
        FakeCut();
        for(int i = 0; i < v_nbr; i++){
            if ( ! isFakeCut[i] ){
                for( Cell cur =edge[i].getHead(); cur != null; cur = cur.rightOf()){
                    int to =(  (Integer)(cur.objectOf())  ).intValue();

                    if ( ! isFakeCut[to] ){

                        List fromL = getChild(i, to);
                        List toL = getChild(to, i);

                        if(!isCross(fromL, toL ) ){
                            g.insertE(i, to);
                            scut[i] = true;
                            scut[to] = true;
                            //System.out.println(i + "\t" + to);
                            count++;
                        }
                    }                           //end if to is not fakecut
                }                               //end inner for loop
            }                                   //end if from is not fakecut
        }                                       //end out for loop
        rt.gc();
        return count;
    }

    public void FakeCut() {
        for ( int i = 0; i < v_nbr; i++){
            int AdjacentNbr = edge[i].getListNbr();
            if ( AdjacentNbr <= 2) isFakeCut[i] = true;
            else isFakeCut[i] = false;
        }
    }

    public void SingleConnection(){
        for(int i = 0; i < v_nbr; i++)
            isFakeCut[i] = false;
        for(int j = 0; j < v_nbr; j++){
                if(! scut[j] ){
                        boolean test = true;
                        for( Cell c = edge[j].getHead(); c != null; c = c.rightOf()){
                                int adj = ((Integer)(c.objectOf())).intValue();
                                if (scut[adj]) {
                                        test = false;
                                        break;
                                }
                        }
                        if ( edge[j].getListNbr()==1 ) test = true;
                        isFakeCut[j] = test;
                }
        }
}

public void findNeighbour(){
        boolean identified[] = new boolean[v_nbr];
        for(int i = 0; i < v_nbr; i ++)
                identified[i] = false;

        //VIP_ID is used to keep track the group it belongs to
        int VIP_nbr = g.getVn() + cv_VIP;
        List VIP_ID[] = new List[VIP_nbr];
        for(int m = 0; m < VIP_nbr; m++)
                VIP_ID[m] = new List();

        int gid = 0;

        SingleConnection();

        for(int j = 0; j< v_nbr; j++){
                if ( ( !scut[j]  ) && ( ! identified[j]) && edge[j].getListNbr() != 1 ){

                        boolean visit[] = new boolean[v_nbr];
                        for(int k = 0; k < v_nbr; k++)
                                visit[k] = false;
                        Queue Q = new Queue();
                        Q.Enqueue(new Integer(j));
                        visit[j] = true;

                        while ( Q.getListNbr() != 0){
                                int cur = ((Integer)(Q.Dequeue())).intValue();
                                identified[cur] = true;
                                member[gid].insert(new Integer(cur));

                                if( !scut[cur] ){
                        for(Cell c = edge[cur].getHead(); c != null; c = c.rightOf()){
                                int adj = ((Integer)(c.objectOf())).intValue();

                                if (!visit[adj] ){
                                        Q.Enqueue(new Integer(adj));
                                        visit[adj] = true;
                                }       }       }
                                else{
                                group[gid].insert(new Integer(cur));
                                VIP_ID[g.findIndex(cur)].insert(new Integer(gid));
                        for(Cell c = edge[cur].getHead(); c != null; c = c.rightOf()){
                        int adj = ((Integer)(c.objectOf())).intValue();

                        if (!visit[adj] && isFakeCut[adj] && !identified[adj] ){
                            Q.Enqueue(new Integer(adj));
                            visit[adj] = true;
                        }}
                        }
                    }                                       //end inner while loop
                    gid++;
                }                                       //end out if
            }                                           //end out for loop
            threshold1 = gid;

            //process the node with all links as short cut
            for(int j = 0; j < v_nbr; j++){
                if (scut[j] && !identified[j] ){
                    boolean visit[] = new boolean[v_nbr];
                    for(int k = 0; k < v_nbr; k++)
                        visit[k] = false;

                    Queue Q1 = new Queue();
                    Q1.Enqueue(new Integer(j));
                    visit[j] = true;

                    while ( Q1.getListNbr() != 0){
                        int cur = ((Integer)(Q1.Dequeue())).intValue();
                        member[gid].insert(new Integer(cur));
                        identified[cur] = true;

                        for(Cell c = edge[cur].getHead(); c != null; c = c.rightOf()){
                            int adj = ((Integer)(c.objectOf())).intValue();
                            if( ( !visit[adj] && edge[adj].getListNbr() ==1 )){
                                Q1.Enqueue(new Integer(adj));
                                visit[adj] = true;
                            }
                            else if ( !visit[adj] && scut[adj] && identified[adj]){
                            if ( group[gid].isFound(cur) ==null)
                            group[gid].insert(new Integer(cur));
                            if(VIP_ID[g.findIndex(cur)].isFound(gid) == null)

        VIP_ID[g.findIndex(cur)].insert(new Integer(gid));
                            }
                        }                   //end inner for loop
                    }                               //end while loop
                    gid++;

                }                                   //end if loop;

            }                                       //end out for loop

            threshold2 = gid;


            //test
            int not_count= 0;
        for ( int l1 = 0; l1 < v_nbr; l1++){
                if ( ! identified[l1] ){
                        not_count ++;
                }
        }
        //Weight[][] is the weight of the clustered graph
        int[][] CGWeight = new int[MAX_G][MAX_G];
        for ( int l1 = 0; l1 < MAX_G; l1++)
                for (int l2 = 0; l2 <MAX_G; l2++)
                        CGWeight[l1][l2] = 0;


        for(int l1 = 0; l1 < g.getVn(); l1++){
                if ( VIP_ID[l1].getListNbr() > 1) {
                        for(Cell c = VIP_ID[l1].getHead(); c != null; c = c.rightOf()){
                                int from = ((Integer)(c.objectOf())).intValue();
                                CGWeight[gid][from] += 2;
                                CGWeight[from][gid] += 2;
                                member[from].delete(g.getAtPos(l1));
                                CG.insertE(gid, from);
                                VIP_ID[l1] = new List();
                                VIP_ID[l1].insert(new Integer(gid));
                        }
                        member[gid].insert(new Integer(g.getAtPos(l1)));
                        gid++;
                }
        }

        GroupNbr = gid;

//build clustered graph
int count = 0;
for(Cell c = g.getV(); c != null; c = c.rightOf()){
int orig_from = ((Integer)(c.objectOf())).intValue();
for ( Cell edge_c = edge[orig_from].getHead(); edge_c != null; edge_c = edge_c.rightOf()){
int orig_to =((Integer)(edge_c.objectOf())).intValue();
int adj = g.findIndex (orig_to);
if ( scut[orig_to] ){
for ( Cell from_c = VIP_ID[count].getHead(); from_c != null; from_c = from_c.rightOf()){
int from = ((Integer)(from_c.objectOf())).intValue();
for ( Cell to_c = VIP_ID[adj].getHead(); to_c != null; to_c = to_c.rightOf()){
int to =((Integer)(to_c.objectOf())).intValue();
CGWeight[from][to] ++;
CGWeight[to][from]++;
CG.insertE(from, to);
                                        }
                                }                       //end inner for loop
                        }
                }                                       //end the for edge list
                count ++;
        }                                               //end the outermost for loop

        //correction for the two counts
        for ( int l1 = 0; l1 < gid; l1++)
                for (int l2 = 0; l2 < gid; l2++)
                        CGWeight[l1][l2] /= 2;
                //print Clustered graph
                //System.out.println("Clustered graph is \n");
                System.out.println("total vertex = " + CG.getVn() + " Edge# = " + CG.getEn());
                System.out.println("Thresh1 =" + threshold1 + "  Thresh2 =" + threshold2);

                System.out.println("Clustered graph based on CGWeight matrix is \n");
                for ( int i = 0; i < gid; i ++){
                        System.out.print( i + " : " );
                        for (int j = 0; j < gid; j++){
                                if (CGWeight[i][j] > 0)
                                        System.out.print(" " + j +"/"+ CGWeight[i][j]+ "->");
                        }
                        System.out.println();
                }

                System.out.println(" cluster = " + CG.isConnected());

        }                                               //end findNeighbour()


public void printGroup(){
        int count = 0;
        for(int i = 0; i < v_nbr; i ++){
                if (group[i].getListNbr() != 0 ){
                        for ( Cell c = group[i].getHead(); c != null; c = c.rightOf()){
                                System.out.print( ((Integer)(c.objectOf())).intValue() + "-> ");
                        }
                        System.out.println();
                        count++;
                }
        }
        count = 0;

        int[] membercount = new int[400];
        for(int i = 0; i < 400; i++){
                membercount[i] = 0;
                int maximum = 0;

                System.out.println("Member information");
                for(int i = 0; i < v_nbr; i ++){
                        int mcount = member[i].getListNbr();
                        if (mcount  != 0 ){
                                System.out.print(" i = " + i +  " ( " + mcount + "): " );
                                if ( i < threshold2 ){
                                        membercount[mcount/10]++;
                                        if ( maximum < mcount) maximum = mcount;
                                }

                                for ( Cell c = member[i].getHead(); c != null; c = c.rightOf()){
                                        System.out.print( ((Integer)(c.objectOf())).intValue() + "-> ");
                                }
                                System.out.println();
                                count++;
```

```
				}
			}
			maximum /= 10;
			System.out.println("Total real cluster number = " + threshold1);
			System.out.println("\tmember number" + "\tcount");
			for( int i = 0; i < (maximum+1); i++){
				if (membercount[i] != 0)
					System.out.println("\t" + i + "\t" + membercount[i]);
			}
		}

		public G getCluster(){
			return CG;
		}

		public void printG(){
			g.print();
		}

		public int getThreshold1(){
			return threshold1;
		}

		public int getThreshold2(){
			return threshold2;
		}

		public int getMemberNbr(int i ){
			if ( i >= 0 && i < GroupNbr)
				return member[i].getListNbr();
			else
				return 0;
		}

		public int getGroupNbr(int i ){
			if ( i >= 0 && i < GroupNbr)
				return group[i].getListNbr();
			else
				return 0;
		}


		public G1 getMember(int i){
			if ( i >= 0 && i < GroupNbr){
				G1 subgraph = new G1();
				//all the member are shortcut
				for ( Cell c = member[i].getHead(); c != null; c = c.rightOf()){
					int node = ((Integer)(c.objectOf())).intValue();

					if ( member[i].getListNbr() ==1 ) {
						subgraph.insertV(node);
					}
					else if ( group[i].getListNbr() == member[i].getListNbr()){
					for ( Cell c1 = edge[node].getHead();c1 != null; c1 = c1.rightOf()){
							int adj = ((Integer)(c1.objectOf())).intValue();
							if (member[i].isFound(adj) != null)
```

```
                                        subgraph.insertE(node, adj);
                                }
                        }
                        else if( !scut[node] ){
                        for ( Cell c1 = edge[node].getHead();c1 != null; c1 = c1.rightOf()){
                                        int adj = ((Integer)(c1.objectOf())).intValue();
                                        subgraph.insertE(node, adj);
                                }
                        }                                       //end for;
                }
                int count = 0;
                for (Cell c = subgraph.getV(); c != null; c = c.rightOf()){
                        int node = ((Integer)(c.objectOf())).intValue();
                        subgraph.setName(name[node],count);
                        subgraph.setScut(scut[node],count);
                        count++;
                }
                return subgraph;
        }
        else return null;
}

public void Function(){
        String category[] = new String[4];
        category[0] ="..\\0430\\CellRole.txt";
        category[1] = "..\\0430\\function.txt";
        category[2] = "..\\0430\\MolecularE.txt";
        category[3] = "..\\0430\\Sub.txt";
        for ( int k = 0; k < 4; k++){
                System.out.println("Category = " + category[k]);
                GetFunction GF = new GetFunction();
                GF.readFunction(category[k]);

                int count[] = new int[GF.getFN()];
                int total = 0;
                for(int j = 0; j < GF.getFN(); j++){
                        count[j] = 0;
                }
                for ( int i = 0; i < v_nbr; i++){
                        if ( scut[i] ){
                                List L = GF.getFunc(name[i]);
                                if ( L != null){
                                for(Cell c = L.getHead(); c != null; c= c.rightOf()){
                                        int func = ((Integer)(c.objectOf())).intValue();
                                                count[func] ++;
                                                total++;
                                        }
                                }
                        }
                }
for (int j = 0; j < GF.getFN(); j++)
        System.out.println("\t" + j + "\t" + GF.getName(j) + "\t" + (float)count[j]/total);
        }
}

}
```

Biobliography

[1] Schwikowski, B., Uetz, P., Fields, S. A Network of Protein-protein Interactions in Yeast *Nature* **18**, 1257-1261 (2000)

[2] Goffeau, A., Barell, B. G., Bussey, H., Davis, R. W., Dujon, B.,Feldmann, H., Galibert, F., Hoheisel, J.D., Jacaq, C., Johnston, M., et.al. *Science* **274**, 563-567 (1996).

[3] Ideker, T., Thorsson, V., Ranish, J.A. et.al Integrated Genomic and Proteomic Analyses of a Systematically Perturbed Metabolic Network *Science* **292**, 929-933(2001)

[4] Jeong, H., Mason, S.P., Barabasi, A.L., Oltvai, Z.N., "Lethality and Centrality in Protein Networks", Nature Vol 411, May 2001, pp41-42

[5] Erdos, P., Renyi, A. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*. **5**, 17-61(1960).

[6] Watts, D.J., Strogatz, S. Collective Dynamics of 'Small World' Networks" , *Nature* **393,** 440-442(1998).

[7] Albert, R., Jeong, H. & Barabasi, A. L. Diameter of the World-Wide Web., *Nature* **400** ,130-131((1999)

[8] Watts, D.J. Small Worlds: The Dynamics of Networks between Order and Randomness. (Princeton University Press, Princeton, New Jersey, 1999)

[9] Adamic, L. The Small World Web. *Proceedings of the European Conf. On Digital Libraries*, 1999.

[10] Albert, R., Jeong, H., & Barabasi, A.L, Diameter of the World-Wide Web, *Nature* **401**, 130-131 (1999)

[11] http://www.mips.biochem.mpg.de/proj/yeast/tables/interaction/physical_interact.html

[12] http://dip.doe-mbi.ucla.edu/

[13] Ito, T. et al. (2000) Toward a protein-protein interaction map of the budding yeast: A comprehensive system to examine two-hybrid interactions in all possible combinations between the yeast proteins. Proc. Natl. Acad. Sci. USA **97**, 1143-1147.

[14] Schwikowski, B., Uetz, P., Fields, S. A Network of Protein-protein Interactions in Yeast *Nature* **18**, 1257-1261 (2000)

[15] Albert, R., Jeong, H., & Barabasi A., L., *Nature* **406**, 378-382(2000).

[16] Jeong, H., Mason, S.P., Barabasi, A.L., Oltvai, Z.N. Lethality and Centrality in Protein Networks *Nature* **411**, 41-42(2001)

[17] Huang, M.L., Eades, P A fully Animated Interactive System for Clustering and Navigating Huge Graphs GD'98, LNCS 1547, S. H. Whitesides Ed. 374-383(1998)

[18] Eades, P., A Heuristic for Graph Drawing, *Congressus Numberantium* **42** ,149-160(1984)

[19] T.Kamada and S. Kawai, An Algorithm for Drawing General Undirected Graphs, *Information Processing Letters* ***30***, 7-15(1989)

[20]Schwikowski, B., Uetz, P., Fields, S. A Network of Protein-protein Interactions in Yeast *Nature* **18**, 1257-1261 (2000)

[21] http://www.proteome.com

[22] Marcotte, E.M., Pellegrini, M., Thompson, M.J., "A combined algorithm for genome-wide prediction of protein function", Nature , Vol 402, 4(1999)pp83-90

[23] Schwikowski, B., Uetz., P., Fields, S., "A Network of Protein-protein Interactions in Yeast", Nature Biotechnology, Vol 18, Dec (2000)pp1257-1261