LUDWIG - MAXIMILIANS UNIVERSITÄT MÜNCHEN
TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Genomorientierte Bioinformatik

Bachelor Thesis

in Bioinformatik

Massive-scale network analysis for automated biomedical synonym resolution

Syeda Tanzeem Haque Charu

Aufgabensteller:	Prof. Hans-Werner Mewes
Betreuer:	Benedikt Wachinger
Abgabedatum:	28.09.2012

Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

28.09.2012

Syeda Tanzeem Haque Charu

Abstract

The ever-growing amount of publications of biomedical journals makes it impossible for an individual to conceive useful information and obtain knowledge. The nascent technology of text mining facilitates the automation of information which results into networks and conversion of information into knowledge. These networks are essential to represent relations between genes, phenotypes, diseases and other entities of interest from the unstructured form of publications. The researcher community usually focuses only on the domain of interest and ignores publications of other domains that might be equally relevant. The research presented here involves an attempt to discover new relations among entities in a massive-scale network and thus provides with new leads on research topics.

This thesis highlights the prospects of utilizing MapReduce in biological networks, in general and a modest search engine for possible new relationships among existing biomedical entities, in particular. The goal of this work is to filter the synonymous entities in the network with the help of a novel local clustering algorithm implemented by MapReduce. MapReduce is a framework to process large amounts of data in parallel and due to the evergrowing massiveness of biological networks we see it justly eligible to utilize MapReduce to such networks. The database of EXCERBT, a text mining tool, was chosen as the standard platform for the research because of its huge number of entities and relations. The whole project was divided into two main categories. First of all, the network was thoroughly analysed on its basic network properties like degree distribution, clustering coefficient and scale-freeness. A few unexpected yet easily dispatchable anomalies were found in the process. Secondly an egocentric local clustering was performed on the whole network. The algorithm designed for this purpose was then applied to cluster synonymous biomedical terms to resolve the intricacy in the network. The main aspect of the analysis and the clustering was to convert the available algorithms in MapReduce algorithms. Apache Hadoop which is a popular free implementation for MapReduce was applied with that end in view. In order to visualize a realistic example of our local clustering algorithm a subnetwork of Parkinson's disease was extracted out of the EXCERBT network. Only genes as entity type were selected for this purpose. This subnetwork was later employed to focus on the biomedical synonym problem related with Parkinson's diesease and eventually towards a proposition for the resolution.

Zusammenfassung

Die stets wachsende Anzahl von biomedizinischen Publikationen macht es unmöglich für einzelne Personen wichtige Informationen zu begreifen und Wissen zu erlangen. Die Text Mining Technologie ermöglicht die Automatisierung der Informationen, was zu Netzwerken führt. Dies führt wiederum zur Umsetzung von Informationen zu Wissen. Solche biomedizinischen Netzwerke sind essentiell um Relationen zwischen Genen, Phenotypen, Erkrankungen und andere Datensätze darzustellen. Normalerweise konzentrieren sich die Forscher und Wissenschaftler nur auf das Thema ihrer Interesse und ignorieren Publikationen anderer Bereiche, die aber auch relevant sein könnten. Die vorliegende Forschungsarbeit hat die Absicht, bisher unbekannte Relationen zwischen Objekten in einem umfangreichen Netzwerk zu finden und dadurch den Forschern neue Hinweise für weitere Forschungen zur Verfügung zu stellen.

Diese Arbeit beleuchtet dabei zunächst im Allgemeinen die Perspektive für die Nutzung von MapReduce in biologischen Netzwerken. Zum anderen wird die konkrete Umsetzung einer einfachen Suchmaschine für potentielle neue Beziehungen zwischen existierenden biomedizinischen Entitäten betrachtet. Das Ziel der Arbeit ist, mithilfe eines neuartigen Algorithmus synonyme Objekte aus einem bestehenden Netzwerk herauszufiltern. Die Implementierung des Algorithmus erfolgte hierbei mit MapReduce, einem Framework für die nebenläufige Berechnung großer Datenmengen, das aufgrund der wachsenden Größe biologischer Netzwerke als geeignet erschien. Die Untersuchung stützte sich auf die Datenbank des Text-Mining-Programms EXCERBT, das über eine genügend große Anzahl an Objekten und Relationen verfügt. Das Gesamtprojekt wurde in zwei Hauptkategorien unterteilt. Zunächst wurde das Netzwerk hinsichtlich der grundlegenden Eigenschaften wie Gradverteilung, Clusterkoeffizient und Skaleninvarianz analysiert. Im Rahmen dessen wurden vereinzelt unerwartete Anomalien entdeckt, die jedoch leicht zu beheben waren. Im nächsten Schritt wurde ein lokales egozentrisches Clustering für das gesamte Netzwerk durchgeführt. Um die Komplexität des Netzwerks zu verringern, wurde der für diesen Zweck entworfene Algorithmus anschließend nochmals für ein erneutes Clustering der synonymen biomedizinischen Ausdrücke angewandt. Der Hauptaspekt der Analyse und des Clustering war die Umwandlung von bereits bestehenden Algorithmen in MapReduce-Algorithmen. Für die Umsetzung stand Apache Hadoop als kostenlose Implementierung von MapReduce zur Verfügung. Als realistisches Beispiel wurde für die Visualisierung der Ergebnisse ein Teilnetzwerk der Parkinson-Krankheit aus dem EXCERBT-Netzwerk extrahiert, wobei lediglich solche Objekte berücksichtigt wurden, die Gen als Objekttyp aufwiesen. Das entstehende Teilnetzwerk wurde schließlich dazu eingesetzt, um die Problematik der biomedizinischen Synonyme in Bezug auf Parkinson hervorzuheben und einen Lösungsvorschlag zu liefern.

Contents

1 Introduction

On February 28, 1953, Francis Crick walked into the Eagle pub in Cambridge, England, and as James Watson later recalled, announced "We have found the secret of life"[1]. They indeed had. Not only that but the two codiscoverers of DNA structure had also laid the cornerstone of a new domain of natural science. Since that day on, biology which is concerned with living organisms, their structure, functions and activities was destined to become an information science. During the last sixty years the concept of information has acquired an outstanding role in biology. Bioinformaticians understand biology in terms of expression of information, execution of programs and interpretation of codes. Algorithms are evolving rapidly, large and heterogenous data are being handled and the capacity seems to know no bounds.

Alone the information of genomic sequence data has increased in such a short time that the notion of data explosion has almost become a cliché. NCBI's Entrez Genome Project site had already cataloged 3,327 genome sequences including prokaryotes and eukaryotes both while the first one (*C. elegans*) was completed not until 1995[2]. Along with the increasing amount of DNA sequencing data there has been a formidable increase in the quantity of data describing how this information is used to implement the functions of the organism. These data are summarized in ever-growing volume of biomedical articles published in different journals. PubMed has a collection of more than 22 million citations[3]. This number is probably hard to comprehend but in 2011 1.4 papers per minute were added to this database on an average and this is an easier number to understand[4].

Yet all this information is not knowledge. One has to perceive, store and retrieve the information with a view to gain knowledge. But without an appropriate tool it is simply impossible for an individual to convert information into knowledge. The nascent technology of text mining promises to automate the information gathering stage by using computers to sift through and scour published articles [1]. The process of text mining can be presented as a pipeline consisting of four phases. It namely comprises the text resources, analyzes the data, then interprets them which finally leads to knowledge (Fig. 1.1). There are three basic types of approaches to text mining in the biomedical domain [2]:

1. **Cooccurrence based system**: This method searches for concepts that occur in the same unit of text, typically a sentence, but sometimes as large as an abstract. Then it posits a relationship between them. For example if BRCA1 and breast cancer occurred in the same sentence it might assume a relationship between breast cancer and the BRCA1 gene.

[1]`http://news.bbc.co.uk/2/hi/science/nature/2804545.stm`, September 26, 2012
[2]`http://www.ncbi.nlm.nih.gov/genomes/static/gpstat.html`, September 26, 2012
[3]`http://www.ncbi.nlm.nih.gov/sites/entrez?db=pubmed&cmd=search&term=1800:2100[dp]`, September 26, 2012
[4]`http://www.nlm.nih.gov/bsd/medline_cit_counts_yr_pub.html`, September 26, 2012

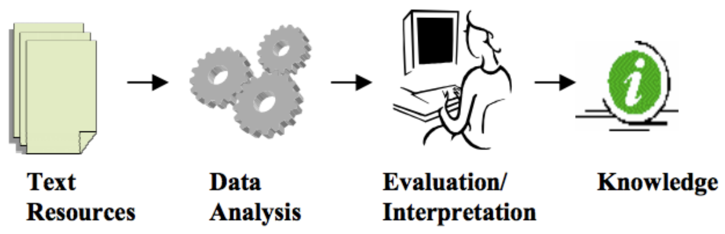


Figure 1.1: The four main steps of text mining. Taken from [3].

2. **Rule based system**: This method works by utilizing general and specific knowledge, particularly the knowledge about language structure and biologically relevant facts stated in the biomedical literature. A simple rule based system might use hardcoded patterns to find explicit statements, for example, ⟨*gene*⟩ plays a role in ⟨*disease*⟩ or ⟨*disease*⟩ is associated with ⟨*gene*⟩. A rule based system might also use sophisticated linguistic and semantic analyses to recognize a wide range of possible ways of making assertions about those classes of things.

3. **Machine learning based system**: This method operates by building classifiers that may operate on any level, from labelling part of speech to choosing syntactic parse trees to classifying full sentences or documents.

In a nutshell text mining establishes one or more relations among the data within a certain domain. The knowledge thus gained through text mining also needs to be stored and represented in such a way that a simple query can lead to the source of the information. A graph is a wide spread prevalent representation of complex relations. The vertices are the entities of interest and the edges the relations between entities. Depending on the texture and the context one can also assign direction and type of relations. This introduces to the possibility of creating a network which can facilitate the modelling of complex dependencies between the entities.

After a certain magnitude of entities and relations even a well planned network can be invaded by disorder and confusion. In order to prevent this situation a network has to be studied, analyzed and clustered if possible. Again here information of multiple theories can be employed from different disciplines like mathematics, computer science or even social sciences. The structure and size of a biomedical network takes well after the global online social networks. It is not only innovative but also inspiring to explore the possibilities of integrating theories from social network analysis into biological networks. In the end it is all about giving the network a comprehensive shape. One might not be able to pigeon-hole the information and knowledge minutely but can definitely count on a sophisticated sorting of the previously prevailing chaos.

Motivation behind the concept

The correlation of complex interactions that lead to a specific phenotype can be apprehended easily with the help of such biomedical networks. The information is already there and so is knowledge. It provides an individual with the idea of what has already been done and

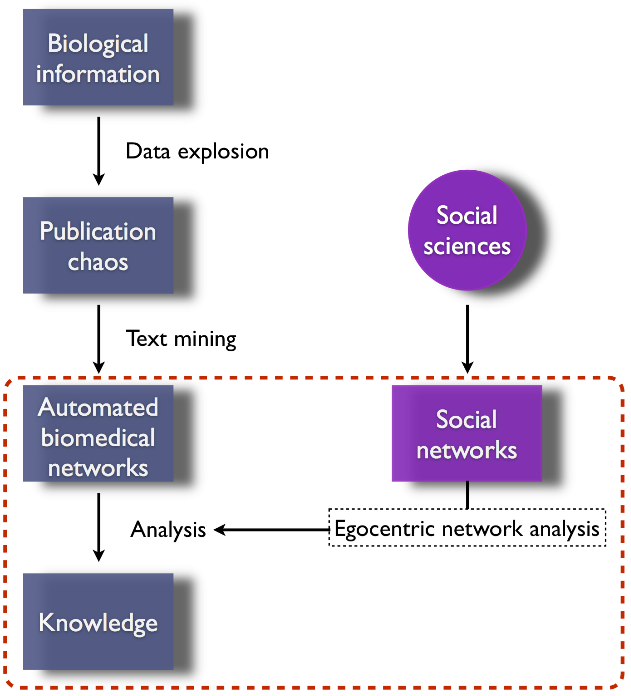


Figure 1.2: A schematic representation of the framework of this thesis. The red dashed rectangle covers the area of the work.

what is yet to be done. One can also study the predicted effects without time consuming and expensive experiments. This thesis enfolds the analysis and clustering of a massive-scale network which was originated by EXCERBT, a text mining tool for biomedical literature. A pattern of our work is sketched in Fig. 1.2. We systematically examine this network and determine its basic properties like degree distribution and clustering coefficients in chapter 3. The main challenge is to run the analysis on this massive-scale network which is beyond the capacity of a single machine. Hence we adjust the conventional algorithm for computation in MapReduce framework (details on MapReduce in chapter 2).

After understanding the network's structure we proceed further for a local clustering in order to utilize the knowledge it provides to the maximum. Here we apply the concept of an egocentric network which has its origins in anthropology. We developed an alogorithm that we implement in MapReduce framework as well. We intend to detect possible new relationships among existing biomedical entities, particularly between genes related to the phenotype Parkinson's disease. It should be mentioned here that the notion of gene, phenotype and the like in this work refers to automated terms in the network. Details follow in chapter 4.

The reason of choosing Parkinson's disease for our work is simply because there is still no cure available for it and there are around 7 to 10 million people living with Parkinson's disease in the world[5]. Parkinson's disease is a slowly progressive disorder that affects movement, muscle control and balance. Part of the disease process develops as cells are destroyed in certain parts of the brain stem, particularly the crescent shaped cell mass known as the

[5]Statistics from the Parkinson's disease foundation, `http://www.pdf.org/en/parkinson_statistics`, September 26, 2012

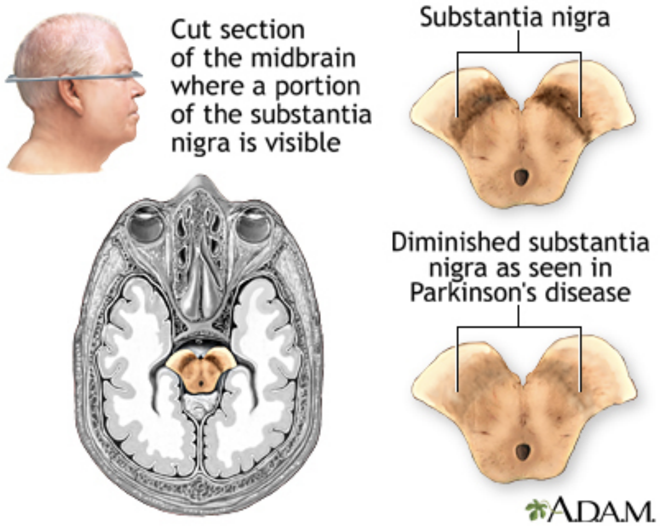


Figure 1.3: Substantia nigra and Parkinson's disease. Figure taken from PubMed Health and copyrighted by A.D.A.M.

substantia nigra (Fig. 1.3). Nerve cells in the substantia nigra send out fibers to tissue located in both sides of the brain. These cells release dopamine, an essential neurotransmitter, that helps control movement and coordination. It is one of the most common nervous system disorders of the elderly. Parkinson's disease most often develops after age 50 which is defined as non familial Parkinson's disease[6]. In such cases it is not subject to natural selection because it does not affect a patient's fitness. In case of familial Parkinson's disease it is possible for patients to develop the disease at a much younger age and pass the defective Parkin gene (mutated PINK1) to their children. This early onset Parkinson's can have an effect on the fitness of the patient and can be subject to natural selection [4][5].

The egocentric local clustering algorithm that we developed in this research relates the genes in pairs according to a specific tie strength (see chapter 4). It suggests gene pairs with strong and weak ties depending on what the researcher is looking for. Besides gene or phenotype this algorithm applies for any type of entity such as protein, metabolite, tissue, species etc.

Additionally we propose a potential usage of the egocentric local clustering algorithm for detecting automated synonymous terms in chapter 5. Also here we experiment on genes related to Parkinson's disease. In this context, a clustering of automated biomedical synonyms should facilitate a better understanding of the correlation of genes with a view to avoid confusion and complexity in the network. This attempt is rather a prospective enrichment to the already gained knowledge than accumulation of further information. Synonym extraction remains solely within the department of text mining where information is still in the process of being converted into knowledge.

[6]Abstracted from `http://www.ncbi.nlm.nih.gov/pubmedhealth/PMH0001762/`, September 26, 2012

2 Materials

2.1 Dataset

There are three datasets that were used in this thesis, namely:

1. EXCERBT
2. Parkinson's disease gene network
3. Protein-protein interation network of *S. cerevisiae*, commonly known as yeast.

The first two were taken for network analysis, local clustering and biomedical synonym resolution by our proposed MapReduce algorithms and the latter, being a familiar network, to validate the algorithms.

2.1.1 EXCERBT

EXCERBT[1], short for *"Extraction of Classified Entities and Relations from Biomedical Text"*, is a text mining resource offered by the Institute of Bioinformatics and Systems Biology (formerly Munich Information Center for Protein Sequences, MIPS) in the Helmholtz Zentrum Munich (IBIS at the Helmholtz Center for Environmental Health, Neuherberg, Germany) [6]. It was developed with a view to reduce the increasing amount of time spent on literature search in the life sciences. It is based on cooccurrence and Semantic Role Labeling (SRL). Semantic role labeling, sometimes also called shallow semantic parsing, is an application in natural language processing (NLP) that identifies the semantic arguments associated with the predicate or verb of a sentence and their classification into their specific roles.

For example:

"BRCA1 encodes a 1,863 amino acid protein with no ascribed function."[2]

Here the task would be mainly to recognize the verb "to encode" representing the relation, the subject "BRCA1" representing the source, the predicate "amino acid" representing the target. This is an important step towards making sense of the meaning of a sentence. A semantic representation of this sort is at a higher level of abstraction than a syntax tree. The passive form of the sentence above is:

"A 1,863 amino acid protein with no ascribed function is encoded by BRCA1."

It has a different syntactic form, but the same semantic roles. Text mining approaches based on cooccurrence alone can handle large databases like PubMed[3] in a modest time but they

[1]`http://mips.helmholtz-muenchen.de/excerbt`, September 26, 2012
[2]`http://www.ncbi.nlm.nih.gov/pubmed/15687549`, September 26, 2012
[3]`http://www.ncbi.nlm.nih.gov/pubmed`, September 26, 2012

cannot extract any specific type of semantic relation. The current implementation of EXCERBT involves the use of SENNA (“Semantic Extraction using a Neural Network Architecture”). SENNA is a fast and accurate neural network based semantic role labeling program that allows a large scale extraction of semantic relations from biomedical literature [7]. Meanwhile, EXCERBT offers the possiblity to submit semantic queries like *"list all phenotypes which are caused by the wrn-gene and show me the sentences supporting the proposed relations"*[4]. Following is a sample table from the EXCERBT database facilitating the reader to grasp the form the database is stored (Tab. 2.1). We postulate that the EXCERBT network is a scale-free one (see 3.2.3 for details on scale-free networks).

Source	Source type	Relation type	Target	Target type	Evidence
brca1	gene	at_activation_act	TLS	gene	pmc:2625440:249:0:senna-2.0:0
lrrk2	gene	at_activation_pas	Microbial	pathway	pubmed:21552986:10:0:senna-2.0:1
dj1	gene	at_inhibition_act	Apoptosis	phenotype	pubmed:11550087:9:0:senna-2.0:0
snca	gene	at_regulation_act	Transgene	phenom	pmc:2722989:91:0:senna-2.0:1

Table 2.1: Stored format of EXCERBT. *"_act"* emphasizes the active form while *"_pas"* the passive form. The term between *"at"* and *"_act"* or *"_pas"* indicates the relation category between the source and the target. There is a sum total of 18 categories representing different verbs in EXCERBT. In this database, every active relation for each tuple of $\langle source, target \rangle$ is entered once again as passive for the tuple $\langle target, source \rangle$ to differentiate between incoming and outgoing edges.

2.1.2 Parkinson’s disease gene network

For the trial run of the local clustering method and as an approach to resolve the problem of automated biomedical synonyms we created a smaller database of Parkinson’s disease from EXCERBT network. We first selected the entities *"Parkinson Disease"*, *"Parkinson disease"* and *"parkinson disease"* as source, then all their targets consisting of type *"gene"* and finally the targets of these targets with type *"gene"* again. Such a network is defined as an *"Egocentric network"* (see 4.1 for details). If considered as an individual network it is basically a hairy ball. Note that the term *"Parkinson diesease"* has the following synonyms[5]:

1. *"parkinson disease"*

2. *"Parkinson’s disease"*

The other term *"Parkinson Disease"* is not entered as a synonym, but has some 12 other disjoint synonyms[6]. Apparently all of these terms should belong to the same set of synonyms. The term *"Parkinson’s disease"* was not considered due to its special character apostrophe (’) which would not play a significant role in any semantic abberation.

We had two versions of this network, a modified one for network analysis and the local clustering and a raw extracted version from EXCERBT for the synonym resolution. Special

[4] `http://www.helmholtz-muenchen.de/en/mips/services/text-mining/index.html`, September 26, 2012

[5] `http://mips.helmholtz-muenchen.de/excerbt/rest/entities/synonyms/Parkinson%20disease`, September 26, 2012

[6] `http://mips.helmholtz-muenchen.de/excerbt/rest/entities/synonyms/Parkinson%20Disease`, September 26, 2012

characters, duplicate entries and self loops were eliminated from the raw network resulting into the modified one (see 3.3.1 for details).

Operation	# Vertices	# Edges
Local clustering	14,906	90,658
Synonym resolution	18,428	110,602

Table 2.2: Number of vertices and edges of the two versions of Parkinson's disease gene network.

2.1.3 Yeast PPI network

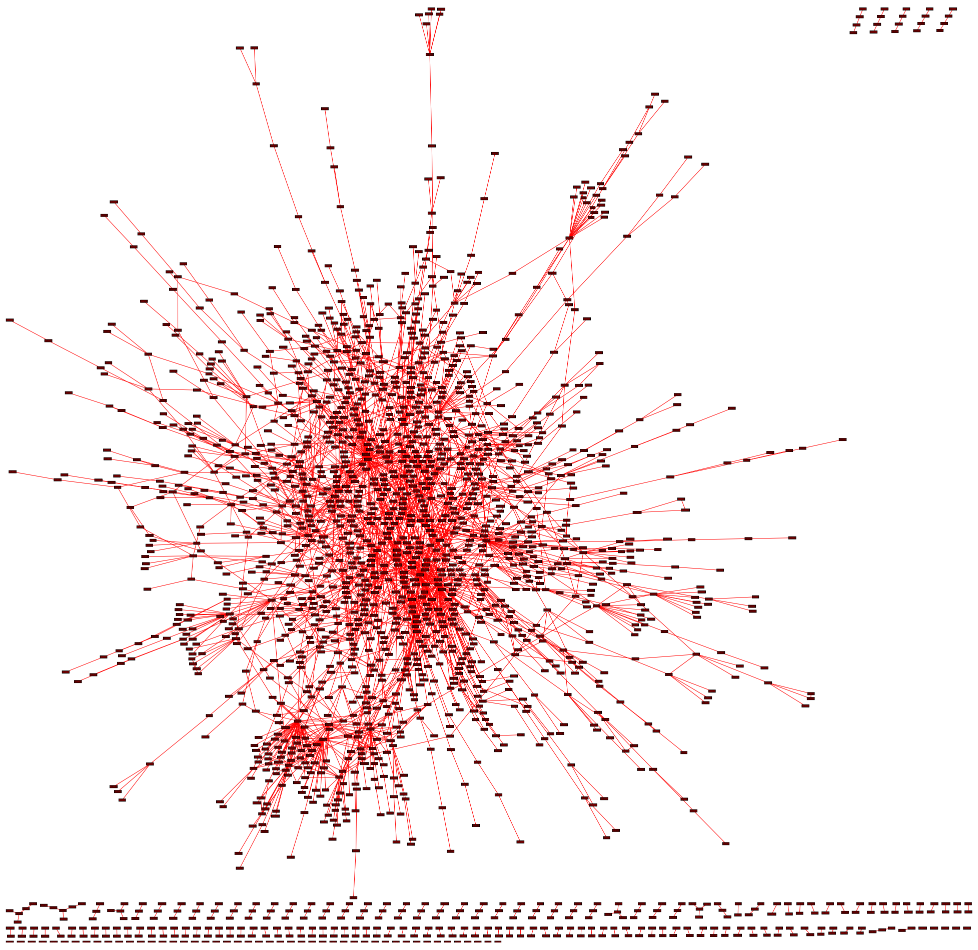


Figure 2.1: Graphical visualization of the Y2H network of yeast interactome used as benchmark.

In order to check the correctness of any novel algorithm one needs to perform tests on a well established and thoroughly investigated dataset. Yeast being the most researched genome in the history of bioinformatics provides with such a dataset of protein-protein interaction. The yeast PPI dataset we chose contains high quality binary interaction information provided by high-throughput yeast two-hybrid (Y2H) screening. It has 2018 proteins and 2929 interactions and is a scale-free network with subtle hierarchy (see 3.2.3 and 3.2.4). We shall use the terminology *Y2H union* for this network henceforth [8]. The database was downloaded from the supplements in Yu *et al.*[7] [8]. The illustration of the Y2H union network is given in Fig. 2.1

[7]`http://interactome.dfci.harvard.edu/S_cerevisiae/index.php?page=download`

which was created by *yEd*, a powerful graph editing application[8], from this database.

2.2 Hardwares and softwares

2.2.1 Hadoop and HDFS

The Institute of Bioinformatics and Systems Biology in the Helmholtz Zentrum Munich provided with a server cloud, *"The EXCERBT cluster"*, containing 20 servers with 160 CPU cores, 480 GB RAM and 168 TB disk space. All the data processing, storage and computation with EXCERBT was performed on this cluster. In our case the Hadoop software framework offers the fundamental architecture for the cloud computing.

The Apache Hadoop software library[9] is a top level Apache project written in the Java programming language, used and still being built by a global community of contributors. This framework allows for the distributed processing of large datasets across clusters of computers. It is designed to scale up from single servers to thousands of machines with local computation and storage and to work with thousands of computational independent computers and petabytes of data. Hadoop was derived from Google's MapReduce (see 2.2.2 for MapReduce) and Google File System (GFS) papers [9].

Hadoop has its own distributed file system, known as HDFS (Hadoop Distributed File System)[10]. It is designed to store very large datasets reliably and to stream those datasets at high bandwidth to user applications. By distributing storage and computation across many servers, the resource can grow with demand while remaining economical at every size [10]. It also offers the facility of *"The JobTracker"* that farms out MapReduce tasks to specific nodes in the cluster, ideally the nodes that have the data, or at least are in the same rack. The JobTracker is a point of failure for the Hadoop MapReduce service. If it goes down, all running jobs are halted.

2.2.2 MapReduce Framework

MapReduce is a programming model for processing large datasets, also originally implemented by Google. It has become a *de facto* standard for parallel computation at terabyte and petabyte scales. MapReduce libraries have been written in many programming languages. A popular free implementation is Apache Hadoop.

As the name itself suggests, the MapReduce computing paradigm works in basically two steps: Map and Reduce. The input and intermediate data are stored in tuples of $\langle key, value\rangle$. The computation proceeds in rounds. Each round is split into three consecutive phases [11] [12]:

1. **Map:** In the map phase the input is processed one tuple at a time. The input data is then split into even number of chunks. This allows different tuples to be processed by different machines and creates an opportunity for massive parallelization. Each machine performing the map operation, also known as a mapper, emits a sequence of $\langle key, value\rangle$ pairs which are then passed on to the shuffle phase.

[8] `http://www.yworks.com/en/index.html`, September 26, 2012
[9] `http://wiki.apache.org/hadoop`, September 26, 2012
[10] `http://hadoop.apache.org/hdfs`, September 26, 2012

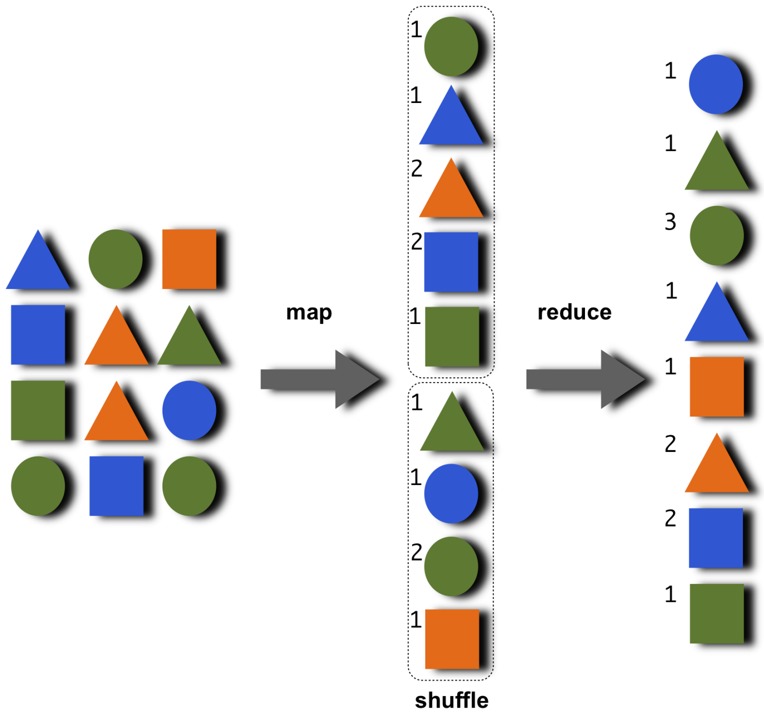


Figure 2.2: Shape counter by MapReduce. All the shapes are splitted into two chunks here, each chunk containing the shape as *key* and the number of shape in that chunk as *value*. Then the chunks are synchronized in the shuffle phase and transferred to the reducer which then counts the real number of each shape by combining them.

2. **Shuffle**: This is the synchronization phase. In this step, the MapReduce infrastructure collects all of the tuples emitted by the mappers, aggregates the tuples with the same *key* together and sends them to the same physical machine.

3. **Reduce**: Finally each key, along with all the values associated with it, is processed together during the reduce phase. The operations on data with one key are independent of data with a different key and can be processed in parallel by different machines.

Fig. 2.2 depicts the phases in a MapReduce procedure[11].

2.2.3 HiveQL

Apache Hive is a data warehouse system for Hadoop that facilitates easy data accumulation, ad-hoc queries, and the analysis of large datasets stored in Hadoop compatible file systems. It was originally developed by Facabook. Among other things it supports a simple SQL like query language that enables users familiar with SQL to query the data. A compiler translates hiveQL statements into a directed acyclic graph of MapReduce jobs, which are then submitted to HDFS for execution [13]. Programmers who are familiar with the MapReduce framework can plug in their custom mappers and reducers to carry out more sophisticated analysis that may not be supported by the built in capabilities of the language[12]. We used hiveQL to analyze EXCERBT on its basic properties like number of vertices, edges, degrees etc. Mainly because it

[11]Adapted from `http://www.gridgainsystems.com`, September 26, 2012
[12]`http://hive.apache.org`, September 26, 2012

is scalable with more machines added dynamically to the Hadoop cluster and can be extended with MapReduce framework and user defined table functions.

2.2.4 Validation program: JUNG framework

The JUNG (Java Universal Network/Graph) Framework is a free, open source software library for graph modeling written in Java. JUNG based applications can make use of the extensive built in capabilities of the Java Application Programming Interface (API), as well as those of other existing third party Java libraries [14]. The framework comes with a number of algorithms from graph theory, data mining and social network analysis such as routines for clustering, decomposition, optimization, random graph generation, statistical analysis, and calculation of network distances, flows, and importance measures (centrality, PageRank, HITS, etc.)[13]. Sequential algorithms for graph theory are already well established and the results can be considered as standard. Since the JUNG framework works sequentially on one single machine we have applied it on the dataset of Yeast PPI network to validate our MapReduce algorithms. We checked whether our proposed alogorithms render accurate results by comparing them with JUNG framework outputs.

[13]`http://jung.sourceforge.net`, September 26, 2012

3 Network Analysis

A network, regardless of its size, needs to be characterized so that it can be compared with any other network. Not only that but one should also have a concrete idea about the basic properties of the network beforehand in order to execute an operation as basic as clustering. There are some elementary measures and models of networks that help the researchers to understand whether their algorithm might even apply for the network under investigation. In this chapter we revised these elementary measures and models of networks in order to analyze the EXCERBT and parkinson disease gene networks.

3.1 Elementary network measures

There are a few basic network measures which describe the properties of a network to a great extent. In the following we define the two most important measures that we further use to analyze the EXCERBT network, namely degree distribution and clustering coefficient. Let $G = (V, E)$ be an unweighted, undirected, non-empty graph. V is the set of vertices, E is the set of edges. Let $n = |V|$, $m = |E|$ and $\Gamma(v)$ be the set of neighbors of a vertex v, i.e. $\Gamma(v) = \{w \in V | (v, w) \in E\}$.

3.1.1 Degree distribution

In graph theory, the degree of a node in a graph is the number of edges incident to the vertex. Or in other words, it is the number of neighbors of the respective node. In this work, the degree of a node v is denoted by the term $d_G(v)$. According to the definition above, $d_G(v) = |\Gamma(v)|$.

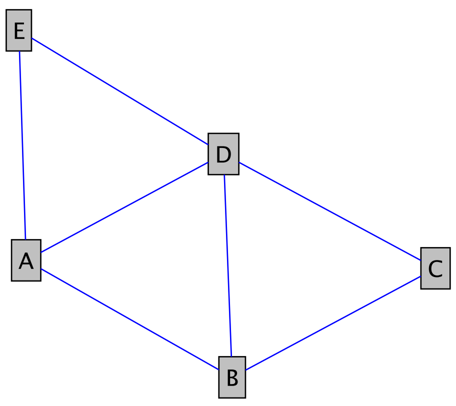
In directed networks (Fig. 3.1b) there is an incoming degree, $d_G(v)^-$, which denotes the number of edges that point to a vertex and an outgoing degree, $d_G(v)^+$, which denotes the number of edges that start from it. A vertex with $d_G(v)^- = 0$ is called a source while a vertex with $d_G(v)^+ = 0$ is called a sink [15].

The degrees of the vertices in the graph in Fig. 3.1 are given in table 3.1:

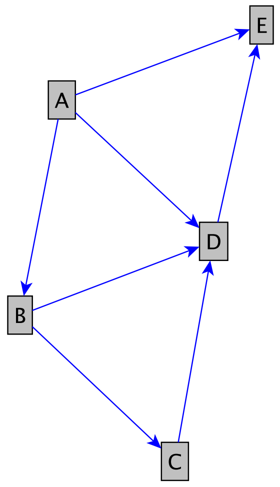
Vertex	Degree($d_G(v)$)	Indegree($d_G(v)^-$)	Outdegree($d_G(v)^+$)
A	3	0	3
B	3	1	2
C	2	1	1
D	4	3	1
E	2	2	0

Table 3.1: Degrees of the vertices in the graph in Fig. 3.1, both as directed and undirected graphs. Here A is a source and$E$ is a sink, if considered in a directed graph.

The degree distribution, $P(d_G)$, is the probability distribution of these degrees over the whole network. It gives the probability that a selected vertex has exactly d_G edges. $P(d_G)$ is obtained by counting the number of vertices $n(d_G)$ with $d_G = 1, 2, 3, ..., n$ edges and dividing by the total number of vertices n (adapted from [16]).



(a) An undirected graph.



(b) A directed graph.

Figure 3.1: Examples of two graphs.

3.1.2 Clustering coefficient

The clustering coefficient of a vertex in a graph gives the measure of the cliquishness of a typical neighbourhood and therefore a local property of a network. Sometimes it is also called the local clustering coefficient but for simplicity we shall further use the term *clustering coefficient* only. The clustering coefficient, $cc(v)$, for a vertex v is given by the fraction of existing edges between its neighbors divided by the number of edges that could possibly exist between them. For each neighbourhood $\Gamma(v)$ there are at most $\frac{d_G(v) \times (d_G(v)-1)}{2}$ edges that could exist among the neighbors of the vertex v. Here $d_G(v)$ is the degree of a vertex v as described above. For a directed graph $\overrightarrow{G} = (V, \overrightarrow{E})$, $(u, w) \in \overrightarrow{E}$ is distinct from $(w, u) \in \overrightarrow{E}$. Thus, the clustering coefficient for directed graphs is given as [17]:

$$cc(v) = \frac{2 \times \frac{1}{2} \times |\{(u, w) \in \overrightarrow{E} | u \in \Gamma(v) \cap w \in \Gamma(v)\}|}{d_G(v) \times (d_G(v) - 1)} \tag{3.1}$$

In case of an undirected graph, the edges (u, w) and (w, u) are considered identical. Therefore the clustering coefficient for undirected graphs is defined as:

$$cc(v) = \frac{2 \times |\{(u, w) \in E | u \in \Gamma(v) \cap w \in \Gamma(v)\}|}{d_G(v) \times (d_G(v) - 1)} \tag{3.2}$$

Let us consider the vertex A in Fig. 3.1. The neighborhood of A is $\Gamma(A) = \{B, D, E\}$.

There are two edges amongst its neighbors, namely between the vertices D and E and the vertices B and D. As a vertex in an undirected graph the clustering coefficient of A is:

$$cc(A) = \frac{2 \times 1}{d_G(A) \times (d_G(A) - 1)} = \frac{2 \times 2}{3 \times 2} = 0.6666...$$

For a directed graph (Fig. 3.1b):

$$cc(A) = \frac{1}{d_G(A) \times (d_G(A) - 1)} = \frac{2}{3 \times 2} = 0.3333...$$

The complete table of the clustering coefficients of the vertices in Fig. 3.1 is given below:

Vertex	$cc(v)$ (undirected)	$cc(v)$ (directed)
A	0.67	0.33
B	0.67	0.33
C	1	0.5
D	0.5	0.25
E	1	0.5

Table 3.2: Clustering coefficients of the vertices in the graph in Fig. 3.1, both as directed and undirected graphs. Here the subgraph with A, D and E form a perfect clique and if isolated, these would have had a clustering coefficient of 1 in an undirected graph.

Another very crucial measure that is derived from both clustering coefficient and degree is the average clustering coefficient of all vertices with d_G edges which we denoted in this work by $C(d_G)$. The average degree $< d_G >$ and the average clustering coefficient $< cc >$ depend on the number of vertices (n) and edges (m). On the contrary, $P(d_G)$ and $C(d_G)$ act as a network's generic features since these are independent of the network's size.

3.2 Network models

All networks can be differentiated into three important models using the two significant generic features ($P(d_G)$ and $C(d_G)$) of a network described in section 3.1. Before analysing any network it is essential to understand the different types of network models and the role of these two features in them.

3.2.1 The small world phenomenon

The small world phenomenon of a network can be described as the situation where most vertices are not neighbors of one another, but can be reached from every other by a small number steps. A small world network is particularly defined to be a network where the typical distance L between two randomly chosen vertices grows proportionally to the logarithm of the number of nodes $n = |V|$ in the network [16]. Social networks, the Wikipedia, gene networks, protein-protein interaction networks are known to exhibit the small world phenomenon.

3.2.2 Random network

A random graph is simple to define. An example of such a random graph is shown in Fig. 3.2a. As a model of a real world network, it has some shortcomings regarding its degree distribution, which is quite unlike those seen in most real world networks. The Erdös–Rényi (ER) model of a random network starts with n number of vertices and connects each pair of vertices $(u, v) \in E$ with an independent probability p with each of the $(n-1)$ other vertices in the graph. The probability $P(d_G(v))$ that a vertex v has exactly the degree $d_G(v)$ is given by the binomial distribution [18][19]:

$$P(d_G(v)) = \binom{n-1}{d_G(v)} p^{d_G(v)} (1-p)^{n-1-d_G(v)} \tag{3.3}$$

The degrees follow a Poisson distribution (Fig. 3.3a) in the limit of large n, which indicates that most vertices have approximately the same number of edges. If plotted against d_G, the $C(d_G)$ appears to be a horizontal line (Fig. 3.4a) revealing the independency between the degree and the clustering coefficient.

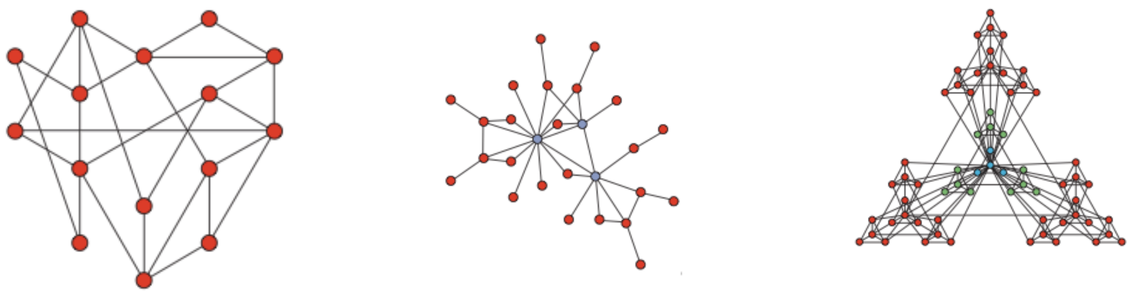

(a) Random network. (b) Scale-free network. (c) Hierarchical network.

Figure 3.2: Examples of three different network models. Adapted from [16].

The average path length of a random networktends to be very small (3.4) equipping it with a small world property [16]:

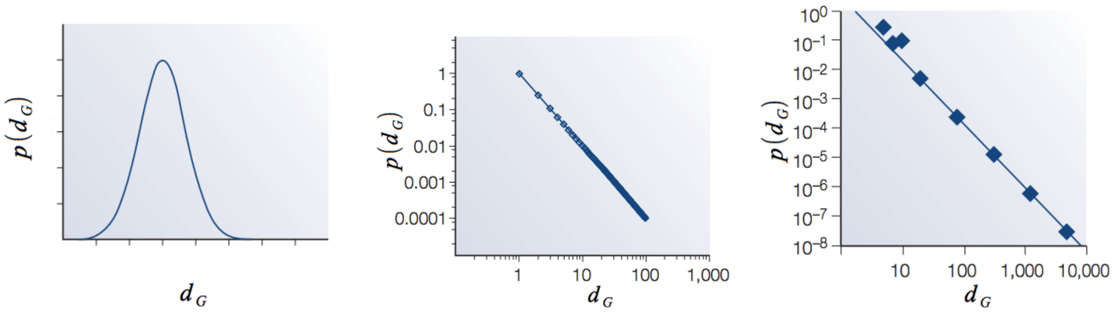
$$L \propto log|V| \tag{3.4}$$

3.2.3 Scale-free network

Most networks in the real world, however, have a highly right-skewed degree distribution. A large majority of vertices have low degree whereas a very few, known as *hubs*, have high degree. Such networks happen to have degree distributions following approximately a power law:

$$P(d_G) \propto d_G{}^{-\gamma} \tag{3.5}$$

Here γ is the degree exponent which remains constant. In most real world networks with scale-free properties γ ranges within the interval of $[2, 3]$. The hubs are not much of a significance if $\gamma > 3$. However a smaller γ indicates quite the opposite. The Barabási-Albert model of a scale-free network (Fig. 3.2b) has a power-law degree distribution that is characterized by the degree exponent $\gamma = 3$. Such distributions are seen as a straight line on a log–log plot (Fig. 3.3b).



(a) Random network. (b) Scale-free network. (c) Hierarchical network.

Figure 3.3: Comparison among the degree distributions of three different network models. Adapted from [16]. (b) and (c) depict log-log plots.

Similar to a random model, $C(d_G)$ is independent of d_G in this model as well. Therefore it does not show any inherent modularity (Fig. 3.4b) [20][16]. Another important feature of a scale-free network is the *ultra* small word phenomenon [21]. The average path length is [16]:

$$L \propto loglog|V| \tag{3.6}$$

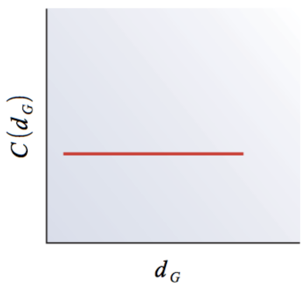
This average path length is significantly smaller than $log|V|$ that characterizes random small world networks (compare Eq. 3.4). These networks play an important role in biological network studies for their structural and dynamical properties. Social networks, the World Wide Web and protein-protein interaction network are scale-free.

3.2.4 Hierarchical network

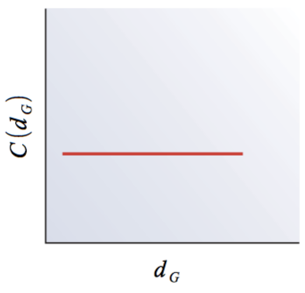
Hierarchical network models are generated by combining clusters with the help of iterative algorithms resulting into the coexistence of modularity, the unique properties of the scale-free topology and the high clustering of the vertices at the same time [22]. In such network models the sparsely connected vertices are part of highly clustered areas and the communication between the different highly clustered neighborhoods are maintained by a few hubs (Fig. 3.2c). A real world example for such a model is shown in Fig. 3.5.

Being part of the scale-free model family, the degree distribution of the hierarchical network model also follows the power law mentioned above (Eq. 3.5). This architecture integrates a scale-free topology with an inherent modular structure by generating a network that has a power-law degree distribution with the following degree exponent [23]:

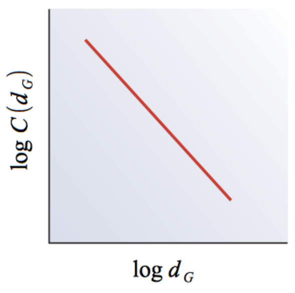
$$\gamma = 1 + \frac{\ln M}{\ln M - 1} \tag{3.7}$$



(a) Random network.



(b) Scale-free network.



(c) Hierarchical network.

Figure 3.4: Dependancy between clustering coefficient and degree of three different network models. Adapted from [16].

Here, M is the replication factor of the network. In Fig. 3.3c, the starting point is a small cluster of four densely linked vertices (the four cyan colored central vertices) and as such the replication factor, $M = 4$ and according to the Eq. 3.7, $\gamma = 1 + \ln 4/\ln 3 = 2.26$.

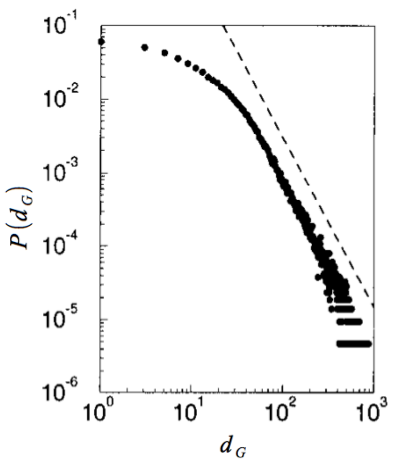


Figure 3.5: Hierarchical network: Log-log plot of degree distribution of the actor collaboration network (also known as the hollywood network) with $n = 212,250$ vertices and average degree $< d_G >= 28.78$. The dashed line has a slope $\gamma_{Actor} = 2.3$. Adapted from [20].

The average clustering coefficient is independent of size of the system and has a value as high as $< cc > \approx 0.6$. The most important signature of hierarchical modularity is the scaling of the clustering coefficient expressed as a function of the degree which is quite the contrary to other scale-free models:

$$C(d_G) \propto d_G{}^{-\beta} \tag{3.8}$$

The exponent β takes the value of 1 in Fig. 3.4c which renders a straight line of slope -1 on a log–log plot.

3.3 Methods

3.3.1 Investigating EXCERBT network and creating test dataset

A prerequisite to any speculation or potential network algorithms is to investigate the network on its fundamental properties. The EXCERBT network was surveyed regarding the following properties with the help of some basic queries by hiveQL (see section 2.2.3):

1. The number of vertices ($n = |V|$) and edges ($m = |E|$) including incoming edges or indegree ($d_G(v)^-$) and outgoing edges or outdegree ($d_G(v)^+$).
2. The number of total relations as a weighted network.
3. The number of relations as a directed network divided in active and passive relations selecting *_act* and *_pas* as relationship type from the database respectively.
4. Total degrees of all vertices and subsequently the degree distribution ($P(d_G)$) (see Fig. 3.9 for result). The EXCERBT network was considered as an undirected and unweighted one in this step.

A modified test set

After observing the results in this step, we created a new test set for further procedures. We observed some junk entities and data anomaly in the EXCERBT database which are nothing but byproducts of text mining. There are also plenty of apparently reduntant entries in this database but these are essential for the semantics, e.g. one should be able to find out the desired same output for both *Protein* and *proteins*. Nonetheless the redundancy needs to be dispatched for transparent results for the computation. The table 3.3 shows an example of such entries.

Case	Source	Source type	Relation type	Target	Target type	Evidence
1.	COX2	gene	at_activation_act	COX-2	gene	pmc:1201567:63:0:senna-2.0:0
	COX-2	gene	at_activation_act	COX2	gene	pmc:1201567:63:0:senna-2.0:0
2.	COX2	gene	at_activation_act	5-@LO	gene	pmc:1794527:20:0:senna-2.0:0
	COX-2	gene	at_activation_act	5-@LO	gene	pmc:1794527:20:0:senna-2.0:0
3.	COX2	gene	at_activation_act	5-LO	gene	pmc:1794527:20:0:senna-2.0:0
	COX-2	gene	at_activation_act	5-LO	gene	pmc:1794527:20:0:senna-2.0:0
4.	COX2	gene	at_regulation_act	5-LO	gene	pmc:1794527:126:0:senna-2.0:1
	COX-2	gene	at_regulation_act	5-LO	gene	pmc:1794527:126:0:senna-2.0:1

Table 3.3: Example of semantically same but physically redundant data in the EXCERBT database. Note that even relation types and evidences are also same for the cases.

COX2 and *COX-2* are the same gene, just the spelling is different. We assume that the case is quite the same for *5-@LO* and *5-LO*. They are not direct synonyms of one another but share a common synonymous term *LOG5*[1]. Without curing the redundancy we have 4 vertices and 8 edges in Tab. 3.3. Therefore we had to prepare a subset of EXCERBT database as our final test dataset by the following way:

[1] `http://mips.helmholtz-muenchen.de/excerbt/rest/entities/synonyms/5-LO`, September 26, 2012
`http://mips.helmholtz-muenchen.de/excerbt/rest/entities/synonyms/5-@LO`

1. Deleting a few junk entities, like: *ans)-*, *cells*, *at(.)*, *result*, *study*, *alle* and *alpha.)-*.
2. Removing special characters like „ :, /, - etc.
3. Converting all entities to lower case. By doing so we ignored the types of both the source and the target entities.
4. Deleting duplicate entries so that every resulting edge is unique.
5. Deleting all self loops.

After this procedure we receive 2 vertices out of Tab. 3.3 namely *cox2* and *5lo* with 1 edge. There are also more than one relation type between two entities abundantly. For example, *COX2* and *5-LO* has the relation type *at_regulation_act* besides the type *at_activation_act* (cases 3 and 4 in Tab. 3.3). One can assign the relation type as unit for weight between two vertices. However we considered the EXCERBT network as an unweighted one. The complete results are presented and discussed to the point in section 3.4.

3.3.2 Calculation of clustering coefficient

It is essential to determine the degree distribution ($P(d_G)$) and the average clustering coeffient ($C(d_G)$) of the EXCERBT network in order to diagnose its charcteristics. In this section we present algorithms with a view to compute the clustering coefficients of the vertices (V) of an undirected graph, $G = (V, E)$. There is also an equivalent way to Eq. 3.2 to view the clustering coefficient. For a pair $(u, w) \in E$ to contribute to the numerator, they must both be connected to each other and to the vertex v itself. The set of vertices $\{u, v, w\}$ must form a triangle in G. A triangle in G is defined as $\{(u, v, w) \subseteq V | (u, v), (v, w), (u, w) \in E\}$. The algorithms presented here to compute clustering coefficient simply determines the numerator of Eq. 3.2 by counting the number of triangles incident on each vertex. Deriving the denominator is not much of a challenge since it can be calculated from the degree of the respective vertex ($d_G(v)$) alone which is counted inevitebly at the beginning of all clustering coefficient algorithms.

Conventional algorithm for clustiering coefficient

We begin by starting with the conventional algorithm for counting the number of triangles in the graph. This algorithm iterates over all of the vertices in the graph. It works by pivoting around each vertex and then checking if an edge exists that will complete any of the resulting 2-paths to form a triangle. This algorithm is equivalent to the one introduced by Watts and Strogatz [17]. The pseudocode is given in Code 1 [11].

Each triangle (u, v, w) is counted 6 times. Once as (u, v, w), (u, w, v), (v, u, w), (v, w, u), (w, u, v) and (w, v, u). Therefore the value is divided by 3 after the triangles count which is then eventually equal to the numerator of the Eq. 3.2. The algorithm runs in a time $O(n {<} d_G {>}^2)$, where n is the number of vertices of the network and $< d_G >^2$ is the average degree squared. A single high degree vertex can lead to a quadratic runtime. Such high degree vertices are often found in real world large graphs. So this algorithm is not practical for massive networks like the EXCERBT network. Even for a small graph the worst case would yield a runtime of $O(n^3)$ where one vertex is connected to every other vertices.

Algorithm 1 Clustering coefficient for (V,E): Sequential

```
1: $C \leftarrow 0; T \leftarrow 0$
2: for $v \in V$ do
3:     for $u \in \Gamma(v)$ do
4:         for $w \in \Gamma(v)$ do
5:             if $(u, w) \in E$ then
6:                 $T \leftarrow T + 1$
7:             end if
8:         end for
9:     end for
10: end for
11: $T \leftarrow T/3$
12: $C \leftarrow T/(|\Gamma(v)| * (|\Gamma(v)| - 1))$
13: return $C$
```

3.3.3 MapReduce approach for clustering coefficient

The algorithm presented above applies for network data fitting into memory of absolutely one single machine. However, any computation on massive graphs has to be executed in more machines and ideally by using parallel algorithms due to their hugeness and storage problem. In this section we demonstrate the adaptation of the sequential algorithm sketched in Code 1 to the MapReduce framework elaborately. For simplicity we have considered the input network as an undirected one. The idea of the algorithm was adapted from [11]. In a nutshell, this attempt lists up all possible edges first, then looks for vertices in the vicinity and determines whether a triangle can be formed or not. The usual concept of searching for triangles from the vertex's point of view is just reversed here.

The original input has to be a simple tab seperated table without heading including all edges amongst the vertices. The table 3.4 shows a sample input for the graph in Fig. 3.6; the heading in the table is for explanatory purpose and should not be included in the original text input. As already mentioned in section 2.2.2, all data are stored in $\langle key; value \rangle$ pairs.

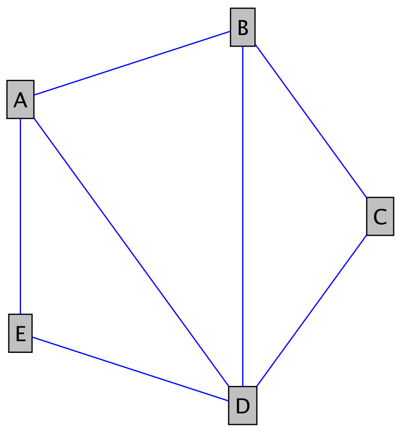


Figure 3.6: The sample input graph for the demonstration of the parallel algorithm by MapReduce.

Source (v) *(key)*	Target (u) *(value)*
A	B
A	D
A	E
B	C
B	D
C	D
D	E

Table 3.4: Tabular input of the sample network in Fig. 3.6 to demonstrate the MapReduce algorithm. The left column (Source) represents the *key* and the right one (Target) the *values*.

Algorithm 2 Clustering coefficient for (V,E): MapReduce

```
1:  **Map 1**: Input: $\langle (u,v); \emptyset \rangle$
2:  emit $\langle u; v \rangle$
3:
4:  **Reduce 1**: Input: $\langle v; S \subseteq \Gamma(v) \rangle$
5:  **for** $u : u \in S$ **do**
6:      emit $\langle (u,v); \$ \rangle$                          ▷ Mark the true edge
7:  **end for**
8:  **for** $(u,w) : u,w \in S$ **do**
9:      emit $\langle (u,w); v \rangle$          ▷ Create a possible triangle out of a possible edge
10: **end for**
11:
12: **Map 2**: Input: $\langle (u,v,\$); \emptyset \rangle, \langle (u,w,v); \emptyset \rangle$
13: emit $\langle (u,w); S \subseteq V \cup \{\$\} \rangle$       ▷ Emit all vertices in the vicinity of the possible edge
14:                                          ▷ For a true edge a special character is included.
15:
16: **Reduce 2**: Input: $\langle (u,w); S \subseteq V \cup \{\$\} \rangle$
17: **if** $\$ \in S$ **then**                                  ▷ If the special character is included
18:     **for** $v \in S \cap V$ **do**
19:         emit $\langle v; 1 \rangle$                           ▷ A true edge => a true triangle
20:     **end for**
21: **end if**
22:
23: **Map 3**: Input: $\langle (v,1); \emptyset \rangle$
24: emit $\langle v; \sum(1) \rangle$
25:
26: **Reduce 3**: Input: $\langle v; \sum(1) \rangle$
27: $i \leftarrow \sum(1)$
28: emit $\langle v; \frac{2 \times i}{(|\Gamma(v)| \times (|\Gamma(v)| - 1))} \rangle$
```

First mapper

The first mapper lists all the vertices and their respective neighbors (Tab. 3.5).

Source (v) *(key)*	Neighbors ($S \subseteq \Gamma(v)$) *(value)*
A	(B, D, E)
B	(A, C, D)
C	(B, D)
D	(A, B, C, E)
E	(A, D)

Table 3.5: The temporary output of the first mapper after processing the input from table 3.4.

First reducer

The reducer works in 2 steps here. In first step it assigns a special character \$ to each true edge $\langle (u, v); \$ \rangle$. This special character is to distinguish the true edges from other possible edges and does not appear any where else in the procedure. In the second step it creates tuples of every two vertices (u, w) contained in the neighbor list of each vertex (v). Then the respective vertex (v) along with its degree $(d_G(v))$ is assigned as *key* to each of the tuples which act as *values* (Tab. 3.6). The degree was not mentioned in Code 2 until the last line where the mathematics is performed. It can be easily calculated in this step by determining the magnitude of the neigh lost. Technically we need to drag along the degree with the vertex from the very beginning. The reducers work in parallel and independent from one another. Being so we would have to risk "losing" the information about the degree otherwise and compute the degree once again in the end. This would be pretty nagging let alone time consuming. A simple concatenation does the trick.

True edges (*key*)	Special character (*value*)	Possible edges (*key*)	Source+degree (*value*)
A, B	\$	B, D	A+3
A, D	\$	B, E	A+3
A, E	\$	D, E	A+3
B, A	\$	A, C	B+3
B, C	\$	A, D	B+3
B, D	\$	C, D	B+3
C, B	\$	B, D	C+2
C, D	\$	A, B	D+4
D, A	\$	A, C	D+4
D, B	\$	A, E	D+4
D, C	\$	B, C	D+4
D, E	\$	B, E	D+4
E, A	\$	C, E	D+4
E, D	\$	A, D	E+2

Table 3.6: The intermediate output of the first reducer after processing the temporary output of the first mapper from table 3.5.

Second mapper

Edge (u, w) (*key*)	Sources (v) (*value*)	Edge (u, w) (*key*)	Sources (v) (*value*)
A, B	($, D+4)	C, D	($, B+3)
A, C	(B+3, D+4)	C, E	(D+4)
A, D	($, B+3, E+2)	D, A	($)
A, E	($, D+4)	D, B	($)
B, A	($)	D, C	($)
B, C	($, D+4)	D, E	($, A+3)
B, D	($, A+3, C+2)	E, A	($)
B, E	(A+3, D+4)	E, D	($)
C, B	($)		

Table 3.7: The temporary output of the second mapper after processing the first reducer output from table 3.6. The highlights are explained in the second reducer step.

In this step the Map process lists up all the *values* from the output of the first reducer according to the respective *keys* (Tab. 3.7).

Second reducer

Source+degree (*key*)	Counts (*value*)
D+4	1
B+3	1
E+2	1
D+4	1
D+4	1
A+3	1
C+2	1
B+3	1
A+3	1

Table 3.8: The intermediate output of the second reducer after processing the temporary output of the second mapper from table 3.7.

The second reducer performs probably the most important task in the whole procedure namely counting the triangles. There are three possible cases in this step:

1. The reducer ignores the case where the size of the *value* list delivered by the second mapper is 1 in spite of the special character $. No triangles can be formed in this case.
2. The reducer also ignores the case if the list contains more than one element but no $. Such examples are highlighted red in table 3.7.
3. Triangles are counted only and only if the list size is greater than 1 and the special character $ is included in it. Entries highlighted blue in table 3.7 represent this case.

If the command for counting triangles is true (the last case listed above), the reducer executes the following actions:

1. Capture each element except $ from the *value* list.
2. Recast it as *key* for the output.
3. Assign 1 as *value* to the *key*, meaning 1 triangle has been counted.

The output is given in table 3.8.

Third mapper

This map process sums up all the 1s of the same *key* and sends them to the third reducer for the final calculation. The sum of the 1s are nothing else than the number of triangles counted per vertex.

Source+degree (*key*)	List of 1s	#Triangles (*value*)
A+3	$\langle 1,1\rangle$	2
B+3	$\langle 1,1\rangle$	2
C+2	$\langle 1\rangle$	1
D+4	$\langle 1,1,1\rangle$	3
E+2	$\langle 1\rangle$	1

Table 3.9: The temporary output of the third mapper after processing the second reducer output from table 3.8.

Third reducer

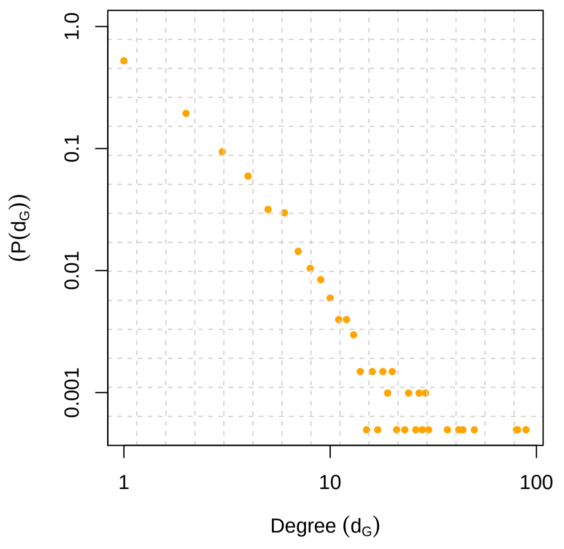
On receival of the temporary output from the third mapper, the third reducer splits the degree $(d_G(v))$ away from the *key* (vertex (v)), fetches its *value* that stands for the respective triangle count and calculate the clustering coefficient by using the Eq. 3.2. The degree contributes to the denominator and the triangle count to the numerator. The table 3.10 presents the clustering coefficients of the vertices of the graph in Fig. 3.6 which is basically the same as the one in Fig. 3.1a. The reducer output of table 3.10 is also same as the clustering coefficients shown in the second column of table 3.2.

Vertex (v)	Degree $d_G(v)$	#Triangles	Clustering coefficient $cc(v)$
A	3	2	0.67
B	3	2	0.67
C	2	1	1
D	4	3	0.5
E	2	1	1

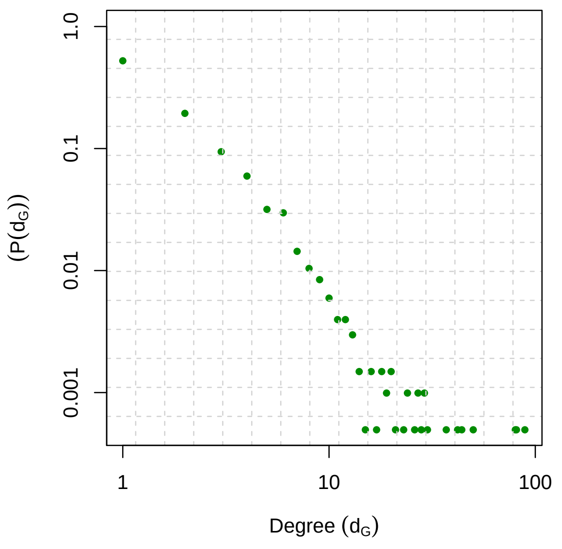
Table 3.10: The final output of the third reducer after processing the temporary output of the third mapper from table 3.9.

3.3.4 Validation

As stated in prevoius chapter (2.1.3) we took the Y2H union network of yeast interactome as benchmark. We applied our algorithm for MapReduce framework and then the relevant algorithms provided by the JUNG framework (2.2.4 to the Y2H union network. We determined its degrees and the clustering coefficients. Then we compared both of the outputs by plotting them.

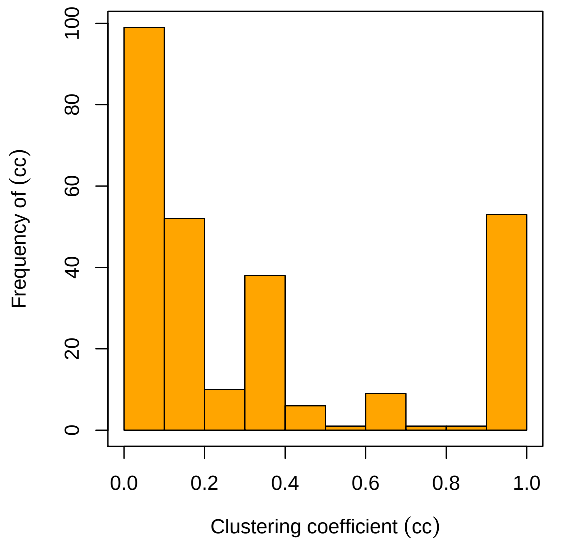


(a) Degrees computed by sequential algorithm (1).

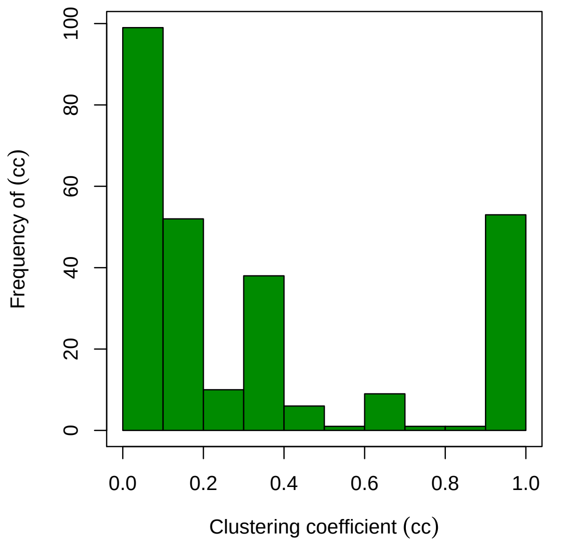


(b) Degrees derived from the first reducer of MapReduce Framework (2).

Figure 3.7: Comparison of the probability distribution of degrees of Y2H union network of yeast interactome computed by the two algorithms.



(a) Clustering coefficient computed by sequential algorithm (1).



(b) Clustering coefficient computed by MapReduce algorithm (2).

Figure 3.8: Comparison of the clustering coefficients of Y2H union network of yeast interactome computed by two algorithms.

Comparison of degree distribution

We plotted the degrees as probability distribution $P(d_G)$ in a log-log plot. Fig. 3.7a and Fig. 3.7b illustrate the degrees computed by JUNG and MapReduce frameworks respectively. Both of plots show exactly the same result.

Comparison of clustering coefficients

To compare the outputs for clustering coefficients we plotted them also as histogram. We omitted the entries with a clustering coefficient of 0.0. Fig. 3.8a illustrates the output by JUNG and Fig. 3.8b the output by MapReduce. Both of the histograms are visibly analog to each other, quite the same as in case of degree distribution.

After observing absolutely no differences in the results we approved of the proposed algorithm for MapReduce framework.

3.3.5 Application of MapReduce algorithm

Finally we applied our MapReduce algorithm to determine degrees and clustering coefficients of the EXCERBT network. Not intending to overload the cluster we conditioned a highest degree of not more than 10,000 for an entity. One can attempt to go for degrees even higher than that but such vertices have no empirical relevance in systems biology. After obtaining the final reducer output we analyzed the behavior of $P(d_G)$ and $C(d_G)$. We used the Eq. 3.5 and Eq. 3.8 for this purpose and calculted the two exponents γ and β. The same procedure was executed on our modified version of Parkinson's disease gene network. We calculated the two exponents γ and β for this network as well.

3.4 Results and discussions

3.4.1 A few numerics concerning EXCERBT network

We found out the following numerics regarding the EXCERBT database after analysing it by hiveQL:

1. Total number of entries in the database regarding source, source types, target, target types, relation tapes and evidences: 1,436,919,650.
2. 718,459,349 rows regarding active relations, i.e. rows with relation type *_act*.
3. 718,459,590 rows regarding passive relations, i.e. rows with relation type *_pas*.
4. 396,548,267 relations or edges considering the network as a weighted one taking the relation type in account.
5. 168,880,702 edges considering the network as an unweighted but directed one comprising 84,440,363 outgoing and 84,440,339 incoming edges.
6. 147,804,853 edges considering it as an unweighted and undirected one.
7. The entity *ans)-* has the highest degree of 179,572. This entity comes from the english article *"an"*.

8. Average degree is: 250.8474 and the median lies at: 17.

The sum of the active and passive relations do not equal the total row count (see Tab. 3.12) in the EXCERBT database:

$$718,459,349 + 718,459,590 = 1,436,918,939 \neq 1,436,919,650$$

There is a difference of 711 rows. We investigated further and found out an anomaly in case of the three entities *Imigrane*, *L 654969* and *Nisis*. Exactly 711 entries (476 entries of active relation type and 235 of passive relation type) regarding these three entities are askewed by one single tab delimiter in the EXCERBT database. The target type is somehow missing which caused a distortion in the entries. As a result, the source type is entered in the column for relation type, the relation type slipped into the column for target and finally the actual target is entered in the taget type column (Tab. 3.11).

Source	Source type	Relation type	Target	Target type	Evidence
Imigrane		metabolite	at_regulation_act	Sensation	pubmed:7935925:1:0:senna-2.0:0
Imigrane		metabolite	at_regulation_act	Thoraces	pubmed:7935925:1:0:senna-2.0:0
L 654969		metabolite	at_activation_act	Activity	pubmed:1964954:3:0:senna-2.0:0
L 654969		metabolite	at_inhibition_act	NB 598	pubmed:8504141:2:0:senna-2.0:0
Nisis		metabolite	at_activation_act	Operon	pubmed:17012392:0:0:senna-2.0:0
Nisis		metabolite	at_activation_act	Immunity	pubmed:17012392:0:0:senna-2.0:0

Table 3.11: Two examples each for the entities: *Imigrane*, *L 654969* and *Nisis* causing the anomaly in the database.

We first assumed that no particular target types were probably assigned to the targets of these entities. But after further investigation we found out the contrary for other sources. The targets possess target types very well. This is not a biological phenomenon nor is it intended for the semantics. It is a casual mistake which can be easily solved.

According to the *handshaking lemma* [24] (Eq. 3.9) the sum of all outdegrees ($d_G{}^+$) should be equal to the sum of all indegrees ($d_G{}^-$) for a balanced directed graph.

$$\sum_{v \in V} d_G(v) = 2|E| \tag{3.9}$$

The EXCERBT network is supposed to be a balanced graph. According to the EXCERBT text mining tool there should be a semantically same incoming edge detected by relation type **_pas* for every outgoing edge detected by **_pas* and vice versa. And yet it was not the case:

$$\sum_{v \in V} d_G(v)^+ = 84,440,363 \neq 84,440,339 = \sum_{v \in V} d_G(v)^-$$

Nevertheless we can ignore this marginal difference comparing it to the massiveness of this network. For further proceedings we ignored the entities *Imigrane*, *L 654969* and *Nisis* and received the following numerics:

1. 349,115 vertices with edges in both directions.
2. 132,576 vertices with only outgoing edges.
3. 100,237 vertices with only incoming edges.

Note that the sum of the vertices while considered with directed edges equals to the number of total vertices in the new set:

$$349,115 + 132,576 + 100,237 = 581,928$$

A brief comparison of the statistics before and after correction is given in table 3.12.

	Before correction	After correction	After modification
# Rows	1,436,919,650	1,362,685,409	
# Vertices	589,222	581,928	424,869
# Edges	147,804,853	140,296,936	90,774,846

Table 3.12: The number of rows, vertices and edges in the EXCERBT database before and after ignoring the three entities *Imigrane*, *L 654969* and *Nisis* and after the final modification.

Henceforth, we considered our new dataset from the EXCERBT database as an undirected and unweighted network and modified it as described in 3.3.1. Our test set now consists of 424,869 vertices and 90,774,846 undirected edges (Tab. 3.12).

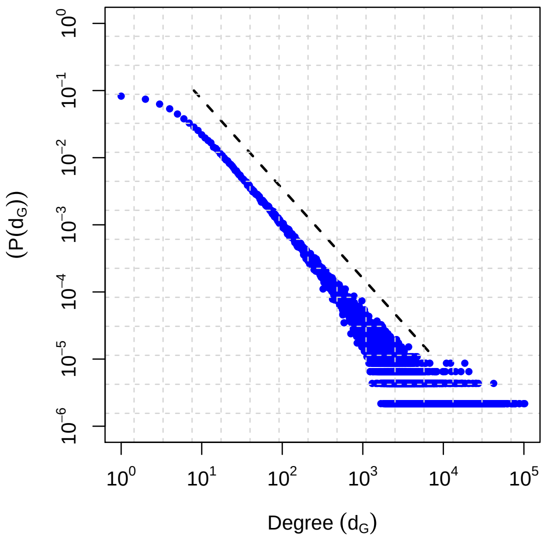


Figure 3.9: Log-log plot of the degree distribution of the EXCERBT network computed by hiveQL which also uses MapReduce. All the vertices are included here. The slope of the dashed line gives the degree exponent γ(=1.1) [20].

3.4.2 Degree distribution of EXCERBT network

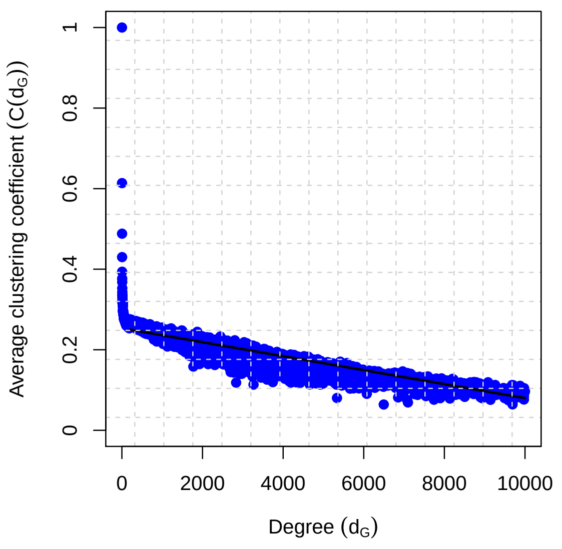
The plot in Fig. 3.9 was derived by hiveQL (2.2.3) for all vertices (424,869 vertices). The probability distribution of degrees (Fig. 3.9) rendered a similar figure to Fig. 3.5. The degree exponent γ that we calculated is (Eq. 3.5):

$$P(d_G) \propto {d_G}^{-\gamma}, \gamma \approx 1.1$$

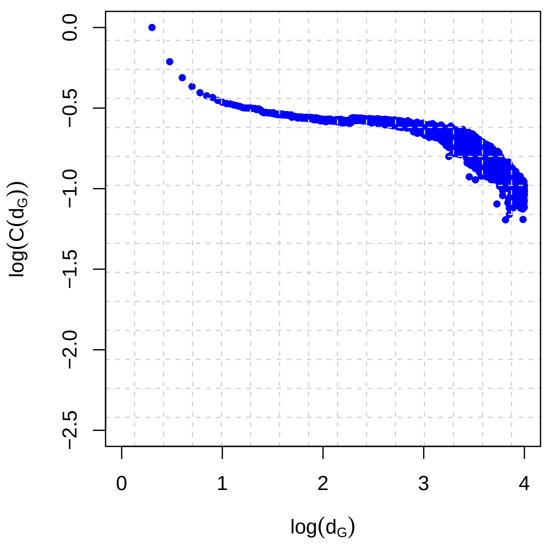
$\gamma = 1.1$ signifies that the hubs play a very significant role in the network and the EXCERBT network definitely is scale-free. In order to define whether it even exhibits modularity or hierarchy, we have to move along on checking the clustering coefficients.

3.4.3 Clustering coefficients of EXCERBT network

It took all together approximately 7 hours for the MapReduce steps to calculate the clustering coefficients. The first plotting of the average clustering coefficient against the degrees (Fig. 3.10a) shows almost a straight line similar to random (Fig. 3.4a) and scale-free (Fig. 3.4b) networks. Since we obtained a slope for the log-log plot of degree distribution (Fig. 3.9) the possibility of the EXCERBT network being a random one can be eliminated (compare with Fig. 3.3b). Yet we logged the both parameters and plotted them to examine the possibility of any hierarchy (Fig. 3.10b). But what was rendered is nothing similar to a hierarchical network (Fig. 3.4c). Hence we could not calculate the $C(d_G)$ and β from Eq. 3.8.



(a) Average clustering coefficient of all vertices with d_G edges plotted against the d_G.



(b) Log of average clustering coefficient of all vertices with d_G edges plotted against the log value of d_G.

Figure 3.10: Behavior of the average clustering coeffiecient against the respective degree in the EXCERBT network. The plot on the left exhibits the scale-freeness of the EXCERBT network whereas the plot on the right cancels the property of any hierarchy.

3.4.4 Degree distribution and clustering coefficients of Parkinson's disease gene network

The computation in MapReduce with the Parkinson's disease gene network took only 5 minutes while the JDK *(Java Development Kit)* on local computer threw a java heap space error. On receiving the results we first plotted the degree distribution on log-log axes and obtained a slope like the EXCERBT network. It is shown in Fig. 3.11. Consequently we calculated the γ and found the following (Eq. 3.5):

$$P(d_G) \propto {d_G}^{-\gamma}, \gamma \approx 1.7$$

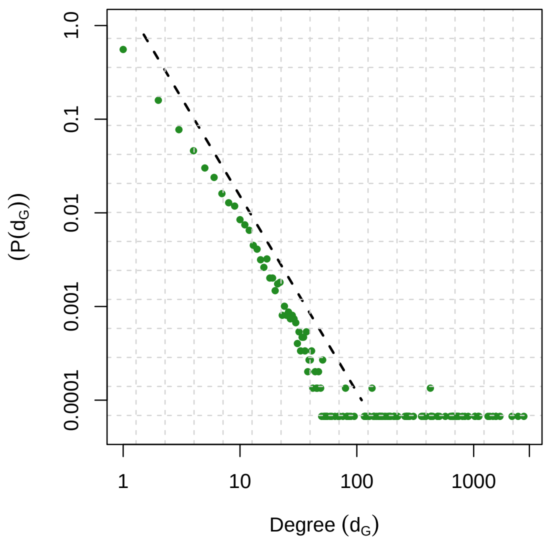
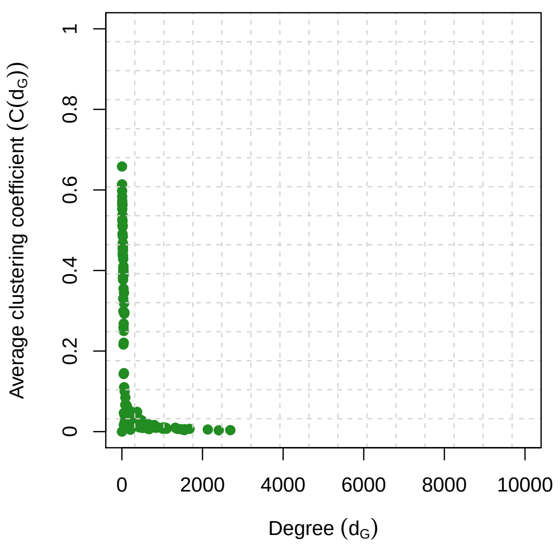


Figure 3.11: Log-log plot of the degree distribution of the Parkinson's disease gene network computed by MapReduce framework. The degree exponent $\gamma = 1.7$ is represented as the slope of the dashed line.

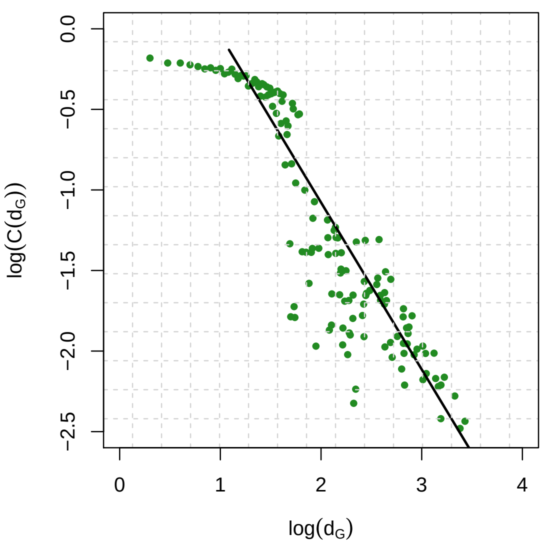
1.7 is a small value for γ and states a scale-free character of the Parkinson's disease gene network. We went further into investigating the architecture. We plotted the $C(d_G)$ against the respective d_G. In contrast to the EXCERBT network we found no straight line which implies that there is very well a dependancy between the two parameters (compare Fig. 3.12a & Fig. 3.10a) . Therefore we logged the parameters and plotted them again (Fig. 3.12b). This time we could indentify a clear slope in the plot and found out the relation between $C(d_G)$ and d_G (Eq. 3.8):

$$C(d_G) \propto {d_G}^{-\beta}, \beta \approx 1.4$$

Although it was mentioned that the β in hierarchical network should be 1, in this case we still can argue that β in this network being 1.4 is approximately 1 and it has a hierarchical architecture. The highest degree found in this network is 2960 while the average lies at 5.95. Both average and median of clustering coefficients are approximately 0.7 which signifies a well connected network. The entity *parkinson disease* itself had a comparatively low clustering coefficient of 0.1. As already mentioned in the previous chapter (2.1.2) this Parkinson's disease gene network is a hairy ball and in this analysis we used the modified version of the database with 14,901 vertices. The entity *parkinson disease* is the central vertex and has 135 direct neighbors and these neighbors have 14,766 neighbors. In an average the vertices at a path



(a) Average clustering coefficient of all vertices with d_G edges plotted against the d_G.



(b) Log of average clustering coefficient of all vertices with d_G edges plotted against the log value of d_G.

Figure 3.12: Behavior of the average clustering coeffiecient against the respective degree in the Parkinson's disease gene network. The plot on the right side indicates a putative hierarchical architecture.

length 1 from *parkinson disease* has around 109 further connections towards the next depth. So there really is a sort of modularity in this network.

Now that we have determined the scale-freeness of the EXCERBT and parkinson disease gene networks and a modularity in the latter we can proceed with further operations that we want to compute with them. The first step would be to cluster them according to certain properties which follows in the next chapter. As for the MapReduce algorithm it surely can be enhanced further in view of runtime. Nevertheless it enabled a modest analysis regarding degree distribution and clustering coefficients undoubtedly.

4 Egocentric Local Clusterer

In this chapter we present an algorithm for local clustering of a massive scale network like the EXCERBT network. The clustering is strongly local, i.e. global attributes like size or architecture of the whole network do not have any impact on the clustering. To be able to do so we have to consider each and every entity in the EXCERBT network as a potential central vertex of an egocentric network, as many in number as all of the entities. In the following we give the reader a brief introduction to egocentric networks as well as a concept about how we intend to perform a local clustering in such networks. Finally we present a potential utilization of the clustering which could lead to discover new relations between entities.

4.1 An egocentric network

The term *Egocentric network* is intertwined with the study of social network analysis. Social network analysis focuses on the measurement of relationships between people. By quantifying the relationships between people, network analysts can apply models and techniques that are commonly used across the social and natural sciences[1]. There are two distinct approaches to social network analysis, namely - the sociocentric network aprroach and the egocentric network approach [25][26][27]. The sociocentric (whole) network approach emerged from sociology and the egocentric (person) approach from anthropology [28]. The latter traces its roots to A. R. Radcliffe Brown, who is considered as one of the fathers of social network analysis. An egocentric network describes the relations between individuals rather than groups and involves the people that a person (referred to as ego) knows. Thus the personal network of a college student includes parents, grand parents, siblings, cousins, relatives, classmates, book club members, or just plain friends as entities. But the student may have more family relations than the head of the faculty who has less time to maintain those relationships.

Similarly enough we can define individual egocentric networks within any biological network for each of its vertices. In this work, we considered the neighbors and the neighbors' neighbors of a vertex as components of such network. We have chosen a path length of only 2 due to the small world phenomena of the networks used here. A path length > 2 would involve supposedly irrelevant entities to the ego-entity and a path length < 2 would leave out important information. Only one assignment of an ego-vertex is possible in global or hairy ball networks. It is quite the case for the Parkinson's disease gene network we used. Fig. 4.1 represents an egocentric network of the protein *YDL190C* extracted from the yeast PPI network.

Such a set of networks would facilitate to examine the structure, shape and composition of one single vertex regardless of the physical property or the content of the whole network. As a matter of fact, egocentric networks allow the membership of one vertex in several net-

[1]Taken from the encyclopedia entry on social network analysis by *Bureau of Economic and Business Research* `http://www.bebr.ufl.edu/files/SNA_Encyclopedia_Entry_0.pdf`, September 26, 2012

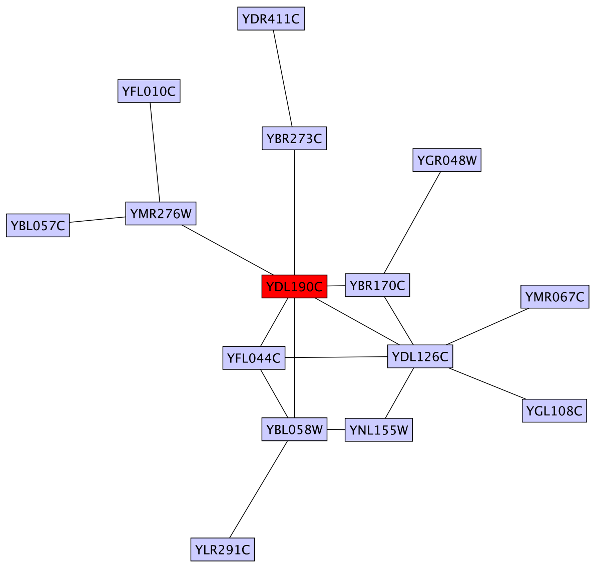


Figure 4.1: An egocentric network of the protein *YDL190C* from the yeast Y2H union network used in this work.

works. By investigating an ego-vertex in an egocentric network individually we can determine the types of connections the ego-vertex emits. Thereby we can generalize its features to other homogenous egocentric networks and detect points of inconsistency among heterogenous ones as well.

4.2 Local clustering

A local clustering algorithm is one that finds a cluster containing or near a given vertex without looking at the whole graph. The one that we shall be using in this work is a so called *friend finder* algorithm with weak ties and strong ties. A fictional friendship network about a few characters from J.R.R. Tolkien's *Lord of The Rings* and J.K. Rowling's *Harry Potter* (Fig. 4.2) explains it in a very simple way. Let us suppose:

1. The two great wizards Dumbledore and Gandalf know each other from graduate school.
2. Being Hobbits from the same village Frodo and Pippin know each other from childhood.
3. Harry met Hermione at *Hogwarts* and they are best friends.
4. As the headmaster of *Hogwarts*, Dumbledore knows Harry and Hermione pretty well.
5. Because of his frequent visits to *The Shire* Gandalf came to know Frodo and Pippin.
6. Frodo met Dumbledore while the latter was visiting *The Shire* with Gandalf.
7. Hermione met Gandalf during one of his visits to Dumbledore at *Hogwarts*.
8. Harry, Frodo and Pippin went fishing together once. Since then Harry is friends with them.

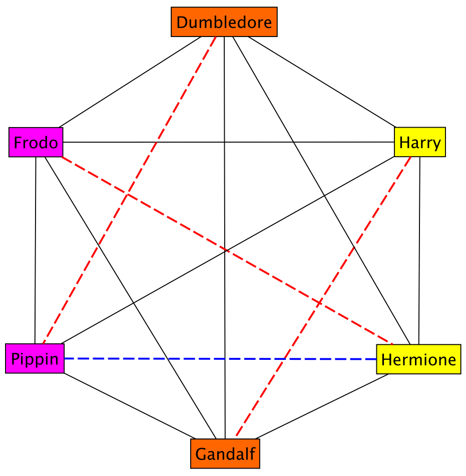


Figure 4.2: Illustration of the idea behind the local clustering algorithm by means of a fictional character graph. The black lines represent the relations that already exist, the red dashed ones the strong ties and the blue dashed ones the weak tie.

These direct relations are shown in Fig. 4.2 by black lines.

The principle of *triadic closure* states that [29]:

> "If two people in a social network have a friend in common, then there is an increased likelihood that they will become friends themselves at some point in the future".

Now let us notice the following facts:

1. Harry might know Gandalf through Dumbledore, Hermione, Frodo and Pippin.
2. Frodo might know Hermione through Harry, Dumbledore and Gandalf.
3. Pippin might know Dumbledore through Frodo, Harry and Gandalf.

In each of the cases above, two persons who do not have any direct relation yet, might have one through a number of common friends. Most of these common friends are also friends with each other. We mark these relations as *strong ties* and these are sketched as red dashed lines in Fig. 4.2.

The following case:

1. Pippin might know Hermione only through Harry and Gandalf.

Harry and Gandalf, the common friends of Pippin and Hermione, do not know each other yet. In this case a *weak tie* exists between Pippin and Hermione. It is marked as blue dashed line in Fig. 4.2.

One might speculate, why weak ties too and why not strong ties only? It has already been observed that people are more likely to acquire jobs that they learned about through individuals they interact with infrequently rather than their close personal contacts [30]. At this point we are trying to synchronize theories from sociology into systems biology via social network

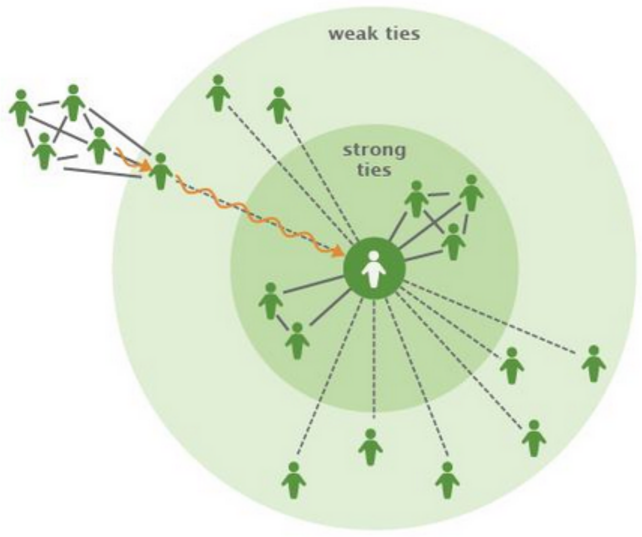


Figure 4.3: The weak ties facilitate information flow from heterogenous clusters [30]. Taken from Facebook.

analysis. Fig. 4.3[2] depicts a model of network illustrating strong and weak ties.

In terms of network architecture, both EXCERBT network and any social network are the same; they are scale-free. Social network is about information flow from persons to persons. Constricting the abstract contents, the EXCERBT network also shows a particular information flow since it was built up on semantic relations. When an entity is connected with two others directly, those entities are also likely to interact with one another. It follows that entities tend to form dense clusters of strong ties which are all connected. These tightly associated clusters are quite small relative to the whole network. Any information that is available to one individual entity spreads quickly to others within the cluster. When it comes to information about future research topics opportunities, it can be hard to find new leads. Weak ties help spread novel information by bridging the gap between clusters of strong tie contacts. In this way one can discover new relations among entities without imposing any artificial cascade effect.

In the following section, we present a *friend finder* algorithm specially for egocentric networks. It has a resemblance with the *triadic closure* principle. A specific score is allocated for each neighbor connection in this algorithm. We have assigned two thresholds for the scores precisely due to the intention of integrating both strong and weak ties in the network. Finally, we implemented the algorithm by MapReduce framework and performed it on EXCERBT, yeast PPI and Parkinson's disease gene networks.

[2]Figure taken from the blog
`https://www.facebook.com/notes/facebook-data-team/rethinking-information-diversity-in-networks/10150503499618859`, September 26, 2012

4.3 Methods

4.3.1 The "friend finder" algorithm

The *friend finder* algorithm here basically works with pairs of direct neighbors of an ego-vertex. It counts how and how often these neighbors are connected with each other, i.e. whether they are connected directly or rather indirectly via another vertex except the ego-vertex in their neighborhood. Then the algorithm groups the pairs according to the strength of their ties [31].

Algorithm 3 Core routine of friend finder algorithm: Sequential

```
function ExpandSeed $(u, S)$:
input: $u$= the user, $S$= the seed
returns: F=the friend suggestions
 1: $G \leftarrow$ GetGroups$(u)$
 2: $F \leftarrow \emptyset$
 3: for each group $g \in G$ do
 4:     for each contact $c \in g, c \notin S$ do
 5:         if $c \notin F$ then
 6:             $F[c] \leftarrow 0$
 7:         end if
 8:         $F[c] \leftarrow^{+}$ UpdateScore$(c, S, g)$
 9:     end for
10: end for
```

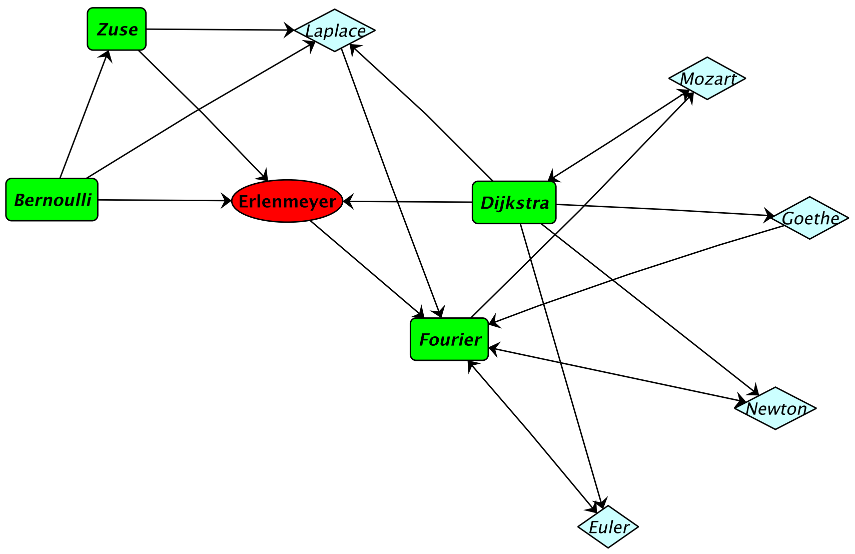


Figure 4.4: A directed graph to demonstrate the technique of the egocentric local clustering. Adapted from [31]

The core routine of the *friend finder* algorithm is explained in the following with the help of a directed graph (Fig. 4.4):

1. Create an egocentric network: get neighbors and neighbors' neighbors for each vertex of interest.
2. Assign a score of 1 for direct contacts netween two neighbors and a score of 0.25 for indirect contacts.
3. Update score: for each pair of direct neighbors (green vertices in Fig. 4.4): sum scores for direct and indirect connections. Scores are given in Tab. 4.1.
4. Normalize the matrix with highest score for each line that corresponds to the neighbor in the matrix. See Tab. 4.2 below for the normalized scores.
5. Define *threshold*$_1$ for strong ties and *threshold*$_2$ for weak ties.
6. Combine neighbors to groups according to thresholds.

[Fourier]	2.00	0.25	0.25
2.00	[Dijkstra]	0.25	0.25
0.25	0.25	[Zuse]	1.25
0.25	0.25	1.25	[Bernoulli]

Table 4.1: Sum of scores of direct and indirect connention for direct neighbor pairs.

[Fourier]	1.00	0.125	0.125
1.00	[Dijkstra]	0.125	0.125
0.125	0.125	[Zuse]	0.625
0.125	0.125	0.625	[Bernoulli]

Table 4.2: Normalized scores from the matrix in Tab. 4.1.

Let us define *threshold*$_1$ = 0.6 and *threshold*$_2$ = 0.2 arbitrarily for the netwrok in Fig. 4.4. Tab. 4.3 shows a classification of neighbor pairs according to the type of ties towards the ego-vertex *Erlenmeyer*.

Pairs	*Fourier*	*Dijkstra*	*Zuse*	*Bernoulli*
Fourier	-	strong	weak	weak
Dijkstra	strong	-	weak	weak
Zuse	weak	weak	-	strong
Bernoulli	weak	weak	strong	-

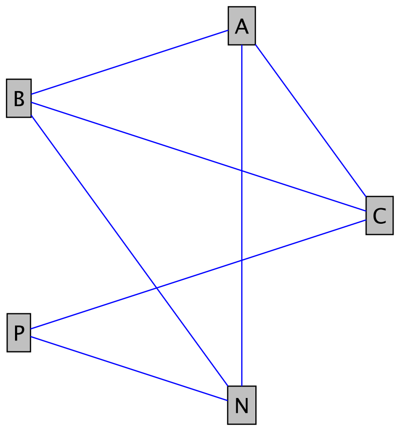
Table 4.3: Neighbor pairs classified by strong and weak ties in two groups.

It might also happen that a neighbor pair is not directly connected at all. In the example above, there is no direct connection between the pair $\langle$*Zuse, Fourier*$\rangle$ nor between $\langle$*Fourier, Dijkstra*$\rangle$. But corresponding the ties in Tab. 4.3 there exist a weak tie and a strong tie respectively. It means their might be very explicit connection between $\langle$*Fourier, Dijkstra*$\rangle$, but neither of them would gain any new information by the newely discovered edge. On the other hand the newly found edge between $\langle$*Zuse, Fourier*$\rangle$ implies a weak relation and it might also be that *Zuse* and *Fourier* have very less in common. Nevertheless this connection would enable a vast information flow from *Fourier* to *Zuse*.

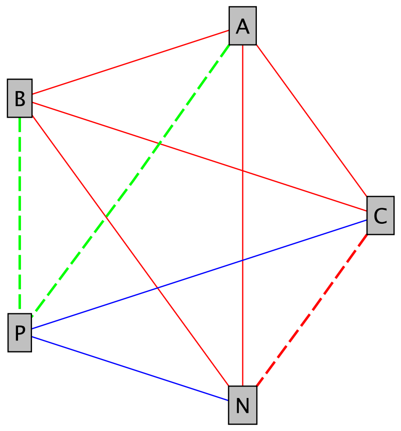
4.3.2 The MapReduce algorithm & its implementation

In this section we show a precise convertion of the sequential algorithm sketched in Code 3 into MapReduce framework. Although the Code 3 applies for directed graphs we have adapted it for an undirected network for the MapReduce algorithm. Our proposed algorithm works in three MapReduce steps. It is a *novel attempt* and its first two steps resemble the Code 2 in essence. The usual definitions already described in the previous chapter (section 3.1) remain the same, i.e. $\Gamma(v)$ stands for the neighbor set of the vertex v.

Similar to the clustering coefficient algorithm for MapReduce framework in the previous chapter the original input has to be a simple tab seperated table without heading including all edges amongst the vertices. A sample input is given in Tab. 4.4. Fig. 4.5 represents the tabular input graphically.



(a) A sample graph to before applying the MapReduce approach.



(b) Result of the MapReduce approach after implementing the algorithm.

Figure 4.5: Demonstration of the proposed local clustering on an egocentric network. The blue edges are the actual connections. The solid red lines on the right graph indicates strong ties which already exist, the dashed red ones and green ones show strong ties and weak ties which may exist respectively.

Source (v) *(key)*	Target (u) *(value)*
A	B
A	C
A	N
B	C
B	N
C	P
P	N

Table 4.4: Tabular input of the sample network in Fig. 4.5a to demonstrate the MapReduce algorithm for *friend finder*. The left column (Source) represents the *key* and the right one (Target) the *values*.

Algorithm 4 Local clustering for (V,E): MapReduce

```
Map 1: Input: $\langle (u,v);\emptyset \rangle$
emit $\langle u;v \rangle$

Reduce 1: Input: $\langle v; S \subseteq \Gamma(v) \rangle$
function LexicoSort $(l_1, l_2)$:                      ▷ Sort a tuple of letters alphabetically
input: $b, a$
returns: $a, b$
for $u : u \in S$ do
    emit $\langle LexicoSort(u,v); v \rangle$          ▷ Mark the true edges by the source v
end for
for $(u,w) : u, w \in S$ do
    emit $\langle LexicoSort(u,w); v \rangle$          ▷ Create all possible tuples of neighbors of v
end for

Map 2: Input: $\langle (u,w,v);\emptyset \rangle$
emit $\langle (u,w); S \subseteq V \cup \{u\} \cup \{w\} \rangle$    ▷ List all vertices to which $(u,w)$
                                                       ▷ could be possible neighbor pair
                                                       ▷ including the mark for true edges if any

Reduce 2: Input: $\langle (u,w); S \subseteq V \cup \{u\} \cup \{w\} \rangle$
if $\{u \cap w\} \in S$ then
    for $v \in S \cap V$ do
        emit $\langle v; (u,w); 1 + 0.25 \times (|S \cap V| - 1); true \rangle$    ▷ Assign 1 for true edge
                                                       ▷ and update score
    end for
else if $\{u \cap w\} \notin S$ then
    for $v \in V$ do
        emit $\langle v; (u,w); 0 + 0.25 \times (|V| - 1); false \rangle$    ▷ Assign 0 for hypothetical edge
                                                       ▷ and update score
    end for
end if

Map 3: Input: $\langle v; (u,w); float; boolean \rangle$
emit $\langle v; T \subseteq \{(u,w); float; boolean\} \rangle$

Reduce 3: Input: $\langle v; T \subseteq \{(u,w); float; boolean\} \rangle$
$F \leftarrow getMax(float)$
for $t \in T$ do
    $newFloat \leftarrow \frac{t(float)}{F}$
end for
emit $\langle v; (u,w); newFloat; boolean \rangle$     ▷ Output with normalized score
```

First mapper

The first mapper lists all the vertices and their respective neighbors (Tab. 4.5).

Source (v) *(key)*	Neighbors ($S \subseteq \Gamma(v)$) *(value)*
A	(B, C, N)
B	(A, C, N)
C	(A, B, P)
P	(C, N)
N	(A, B, P)

Table 4.5: The temporary output of the first mapper after processing the input from Tab. 4.4.

First reducer

The first reducer accomplishes the following tasks:

1. It creates tuples of edges by combining the source vertex with each of its neighbors, which are then obviously true edges.

2. Then it creates tuples of vertices from the neighbor set of the source vertex which stand for possible or hypothetical edges. These might be true edges or not. The reducer does not know it yet.

3. The elements, i.e. vertices in each tuple are emitted alphabetically. If a combination creates a $\langle z, a \rangle$ edge, the output would be $\langle a, z \rangle$.

4. The tuples are assigned as *keys* and source vertices as *values* for the next mapper.

The first reducer output is given in Tab. 4.6.

All edges (*key*)	Source (*value*)	All edges (*key*)	Source (*value*)
A, B	A	C, P	C
A, C	A	A, B	C
A, N	A	A, P	C
B, C	A	B, P	C
B, N	A	C, P	P
C, N	A	N, P	P
A, B	B	C, N	P
B, C	B	A, N	N
B, N	B	B, N	N
A, C	B	N, P	N
A, N	B	A, B	N
C, N	B	A, P	N
A, C	C	B, P	N
B, C	C		

Table 4.6: The intermediate output of the first reducer after processing the temporary output of the first mapper from Tab. 4.5. The lexicographical keys represent all edges including true and hypothetical ones. The true edges marked for one single source vertex are highlighted red.

Second mapper

In this step the Map process lists up all the *values* from the output of the first reducer corresponding to the *keys* (Tab. 4.7).

Edge (u, w) (*key*)	Sources (v) (*value*)	Edge (u, w) (*key*)	Sources (v) (*value*)
A, B	(A, B, C, N)	B, N	(A, B, N)
A, C	(A, B, C)	B, P	(C, N)
A, N	(A, B, N)	C, N	(A, B, P)
A, P	(C, N)	*C, P	(C, P)
B, C	(A, B, C)	*N, P	(P, N)

Table 4.7: The temporary output of the second mapper after processing the first reducer output from Tab. 4.6 The red highlighted vertices in each list indicate that the corresponding key indeed is a true edge.

(*key*)	(*value*)			
Ego-vertex	Neighbor pair	Direct connection between neighbor pair	# Indirect connection between neighbor pair	Score
C	A, B	true	1: via N	1.25
N	A, B	true	1: via C	1.25
B	A, C	true	0	1.0
B	A, N	true	0	1.0
C	A, P	false	1: via N	0.25
N	A, P	false	1: via C	0.25
A	B, C	true	0	1.0
A	B, N	true	0	1.0
C	B, P	false	1: via N	0.25
N	B, P	false	1: via C	0.25
A	C, N	false	2: via B & P	0.5
B	C, N	false	2: via A & P	0.5
P	C, N	false	2: via A & B	0.5

Table 4.8: The intermediate output of the second reducer after processing the temporary output of the second mapper from Tab. 4.7. Except the furthest left column which contains the *keys*, all the other column entries contribute together as *value*. The order of the value output can be user defined.

Second reducer

The second reducer performs the most significant task in the whole procedure. It allocates scores to the neighbor pairs of the ego-vertices regarding their connections. It is important to mention here that the lists *(values)* from Tab. 4.7 contain potential ego-vertices. The technique is as follows:

1. The reducer scans every list *(value)* of a neighbor pair *(key)* for the two contributing vertices that build up the latter.
2. If step 1 returns true we have a case of *true edge*, meaning the neighbor pair is already connected directly (cases highlighted red in Tab. 4.8). In this case the reducer leaves the two contributing vertices alone and takes each of the other vertices left in the list as an ego-vertex.

3. While proceeding step 2, if there are only the two contributing vertices in the list it would indicate that although the certain pair has a direct connection, they do not have any indirect connections. Such neighbor pairs are ignored all together. Such cases are marked with an asterisk in Tab. 4.7.

4. If step 1 returns false we have a *hypothetical edge*. The neighbor pair is not directly connected to each other, rather indirectly via every single vertex contained in the list. In this case the reducer considers every vertex in the list as an ego-vertex.

5. Once detected, an ego-vertex corresponds to a score of 1 or 0 for the neighbor pair, depending on the case of a *true edge* or a *hypothetical edge* respectively.

6. After that the ego-vertex corresponds to a score of 0.25 for the neighbor pair per each other vertex in the list except the ego-vertex itself.

7. The scores for neighbor pair of each ego-vertex are summed up and the reducer is ready for the output (Tab. 4.8).

The ego-vertex is the *key*. Each element of a *value* includes the neighbor pair, the respective score as float and a boolean value which is true for *true edge* and false for *hypothetical edge*.

Third mapper

This mapper brings the *values* to a *key* from the output of the third reducer together.

Ego-vertex *(key)*	List of neighbor pairs with scores *(value)*
A	[((B, C), 1.0, *t*), ((B, N), 1.0, *t*), ((C, N), 0.5, *f*)]
B	[((A, C), 1.0, *t*), ((A, N), 1.0, *t*), ((C, N), 0.5, *f*)]
C	[((A, B), 1.25, *t*), ((A, P), 0.25, *f*), ((B, P), 0.25, *f*)]
N	[((A, B), 1.25, *t*), ((A, P), 0.25, *f*), ((B, P), 0.25, *t*)]
P	[((C, N), 0.5, *f*)]

Table 4.9: The temporary output of the third mapper after processing the second reducer output from Tab. 4.8. *t* stands for a true edge and *f* states that there is no connection between the pair.

Third reducer

The last reducer step normalizes the score among the neighbor pairs of each ego-vertex. This can be done during the third mapping step too. The strength of ties between the neighbor pairs are determined here. We defined *score* ≥ 0.5 as *threshold*$_1$ and $0.2 \geq$ score < 0.5 as *threshold*$_2$ for the sample graph in Fig. 4.5a. The Tab. 4.10 shows the grouping of the neighbor pairs according to the strength oftheir ties..

Ego-vertex	Neighbor pairs with strong tie	Neighbor pairs with weak tie
A	((B, C), 1.0, *t*) ((B, N), 1.0, *t*) ((C, N), 0.5, *f*)	none
B	((A, C), 1.0, *t*) ((A, N), 1.0, *t*) ((C, N), 0.5, *f*)	none
C	((A, B), 1.0, *t*)	((A, P), 0.2, *f*) ((B, P), 0.2, *f*)
N	((A, B), 1.0, *t*)	((A, P), 0.2, *f*) ((B, P), 0.2, *f*)
P	((C, N), 1.0, *f*)	none

Table 4.10: The final output of the third reducer after processing the temporary output of the third mapper from Tab. 4.9. The scores are normalized here.

4.3.3 Application of MapReduce algorithm

Y2H network of yeast interactome

The algorithm was first applied to the respectively smaller Y2H union network of yeast interactome. After creating the local clusters we investigated the interactions of the protein YDL190C as ego-vertex. We isolated YDL190C from the network first and observed the impact of the clustering on it. Then we observed it from the point of view of the whole network and detected the differences. We also examined the role of the egocentric networks in the vicinity of a neighbor pair on its tie strength.

EXCERBT network

We executed the algorithm on EXCERBT network taking only entities with a degree less than 5,000 and there were 341,854 of them. As already described in previous chapter (3.1.2) for each neighborhood of an ego-vertex (v) with degree $d_G(v)$ there exist $\binom{d_G(v)}{2}$ many possible triangles. This number also represents the possible number of neighbor pairs. So alone with 5,000 direct neighbors an ego-vertex will have around 12.5 millions tuples to deal with. For 10,000 direct neighbors the number of tuples will go as high as 50 millions. Hence we proceeded carefully and restricted the degree to 5,000 with a view not to overload *the EXCERBT cluster* massively. The program specially written for MapReduce framework works in such a way that it considers each and every vertex available as an ego-vertex at a time and yet computes in parallel. We integrated a so called *edge tag* in the program which appointed the true edges a "1" and the hypothetical one a "0". Thus we could filter out neighbor pairs with a significant tie score and a potential connection. An extra mapper and reducer were implemented for the assortment.

Parkinson's disease gene network

Finally we applied the algorithm on the modified version of Parkinson's disease gene network consisting of 14,906 vertices and 90,658 edges. We assigned the entity *parkinson disease* as the ego-vertex. In order to investigate the result properly we extracted the tie scores between neighbor pairs for *parkinson disease* from the result obtained after clustering the EXCERBT network. Then we compared the both sets of tie scores to identify any discrepancy. The entity *parkinson disease* has 135 direct neighbors and only 9,045 possible neighbor pairs. So it was visibly a lot easier to deduce any conclusion. We selected one gene from the network and investigated both its available and putative connections above the thresholds.

4.4 Results and discussions

4.4.1 The threshold problem

We still had to ponder over the perfect thresholds to determine. Many hypotheses can be produced and many theories remain on how to define a meaningful threshold and this is still a promlem. We tried to define our threshold for weak ties as the following [31]:

1. Lower threshold in networks with one neighbor pair with many connections and less connections between all other pairs.
2. Higher threshold in networks with several highly connected neighbor pairs.

We plotted the tie scores as histogram and defined the threshold from the score distribution.

4.4.2 Y2H network of yeast interactome

The first try out of the MapReduce *friend finder* algorithm with the Y2H network was very simple. All vertices in this network could be assigned as the ego-vertex. The highest number of neighbor pairs is 3,916 and the respective ego-vertex is YLR291C with the highest degree of 89. We isolated the protein YDL190C from the network with its neighbors and neighbors' neighbors and found out five significant pairs out of 15 (Tab. 4.11). The other 10 pairs had a score of 0.0 each. The ties are shown in Fig. 4.6. The pairs $\langle$YDL126C, YFL044C$\rangle$ and $\langle$YDL126C, YBR170C$\rangle$ exist already and have strong ties. Hence the edges between are depicted by solid red lines. We defined $0.2 \geq$ scores > 0.5 as the weak threshold and scores ≥ 0.5 as the strong threshold in this case.

Neighbor 1	Neighbor 2	Tie score	Tie type
YBL058W	YFL044C	1.0	strong
YBL058W	YDL126C	0.5	strong
YBR170C	YFL044C	0.25	weak
YBR170C	YDL126C	1.0	strong
YFL044C	YDL126C	1.0	strong

Table 4.11: Significant neighbor pairs for YDL190C.

Let us now consider the ego-vertex YDL190C not in a single egocentric network but as a member of the whole Y2H network. At this step we are no longer concentrating ourselves on the ego-vertex itself but on the neighbor groups been suggested for it by our algorithm. The neighbor pairs suggested for YDL190C can also be suggested for any other ego-vertex. So we need to observe the imposed effect of other ego vertices in the vicinity. In this case we examine how the two neighbor pairs $\langle$YBR170C, YFL044C$\rangle$ and $\langle$YBL058W, YDL126C$\rangle$ suggested for YDL190C might be affected by others.

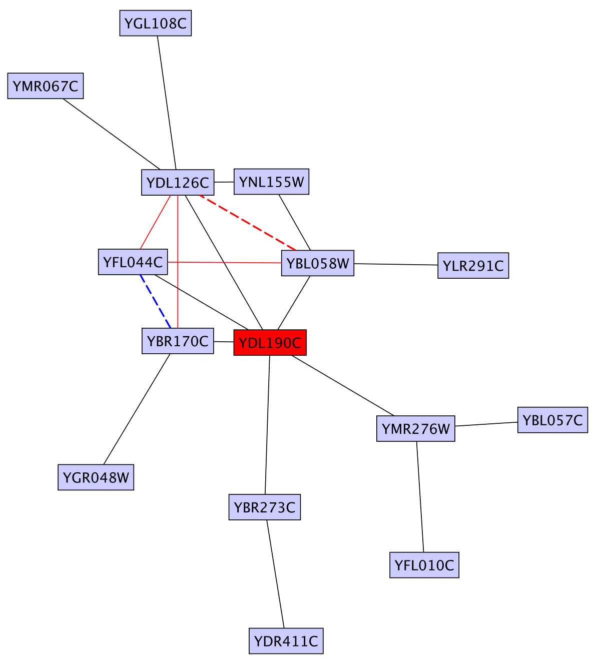


Figure 4.6: Local clustering after isolating the vertex YDL190C from the original Y2H network. The red solid lines show already existing edges with strongs ties between neighbors. The dashed red one shows a possible strong between YDL16C and YBL058W while the blue dashed one a possible weak tie between YFL044C and YBR170C.

⟨YBR170C, YFL044C⟩

This neighbor pair was also suggested by YDL126C with a score of 0.2. This protein is a direct neighbor of YDL190C and belongs to its egocentric network. It has interactions with 6 other proteins and 15 neighbor pairs can be created for it. While considering YDL126C as an ego-vertex 10 of the neighbor pairs got a score of 0.0. The scores of the other 5 pairs are given in Tab. 4.12. Consequently we selected 0.2 to be a weak tie.

Neighbor 1	Neighbor 2	Tie score	Tie type
YNL155W	YFL044C	0.2	weak
YNL155W	YDL190C	0.2	weak
YBR170C	YFL044C	0.2	weak
YBR170C	YDL190C	0.8	strong
YFL044C	YDL190C	1.0	strong

Table 4.12: Significant neighbor pairs for YDL126C. The pair under investigation is highlighted blue.

We scanned the output data of the local clustering for evidence. The pair ⟨YBR170C, YFL044C⟩ was suggested indeed for only YDL190C and YDL126C with a tie score of 0.25 and 0.2 respectively. Therefore we can ascertain that an explicit weak tie exist between this particular pair (Fig. 4.7).

⟨YBL058W, YDL126C⟩

Besides YDL190C this pair was suggested for YFL044C and YNL155W. The former is a direct neighbor of YDL190C and the latter is not. But both of them belong to the egocentric network

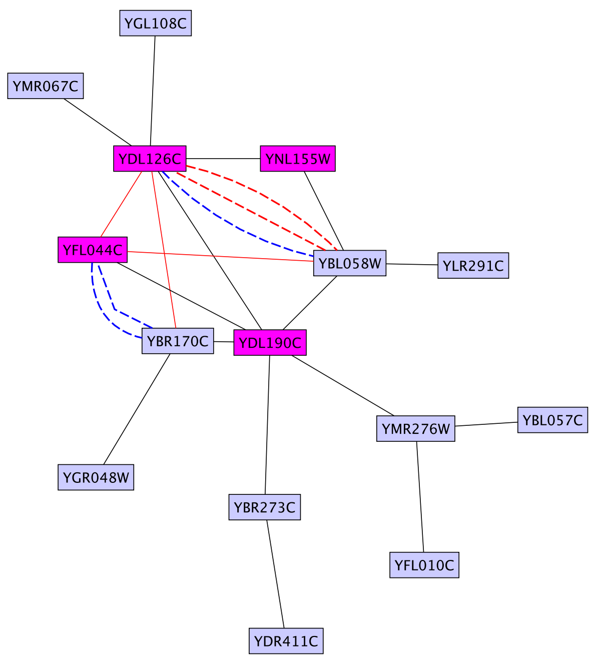


Figure 4.7: Local clustering considering YDL190C within the originial Y2H network. The red solid lines show already existing edges with strongs ties between neighbors. The dashed red one shows a possible strong tie and the blue dashed ones a possible weak tie. The multiple ties are to indicate that the particular neighbor pair was not only suggested for YDL190C but for other ego-vertices (marked in purple rectangles) also. We found an ambivalent tie between YDL126C and YBL058W.

of YDL190C. The pair ⟨YBL058W, YDL126C⟩ got a score of 1.0 assigned by YNL155W and a score of 0.4 by YFL044C. From YDL190C itself it got a score of 0.5. The scores for the egocentric networks of YFL044C and YNL155W are given in Tab. 4.13.

Now we have a problem since the pair under investigation got ambiguous ties. We could assign the average score out of 1.0, 0.5 and 0.4, which is 0.63, for ⟨YBL058W, YDL126C⟩ and argue for a strong tie. At this point we are speculating an edge which is a common member in a set of egocentric networks.

Ego-vertex	Neighbor 1	Neighbor 2	Tie score	Tie type
YFL044C	YBL058W	YDL190C	0.8	strong
	YBL058W	YDL126C	0.4	weak
	YDL190C	YDL126C	1.0	strong
YNL155W	YBL058W	YDL071C	0.0	null
	YBL058W	YDL126C	1.0	strong
	YDL071C	YDL126C	0.0	null

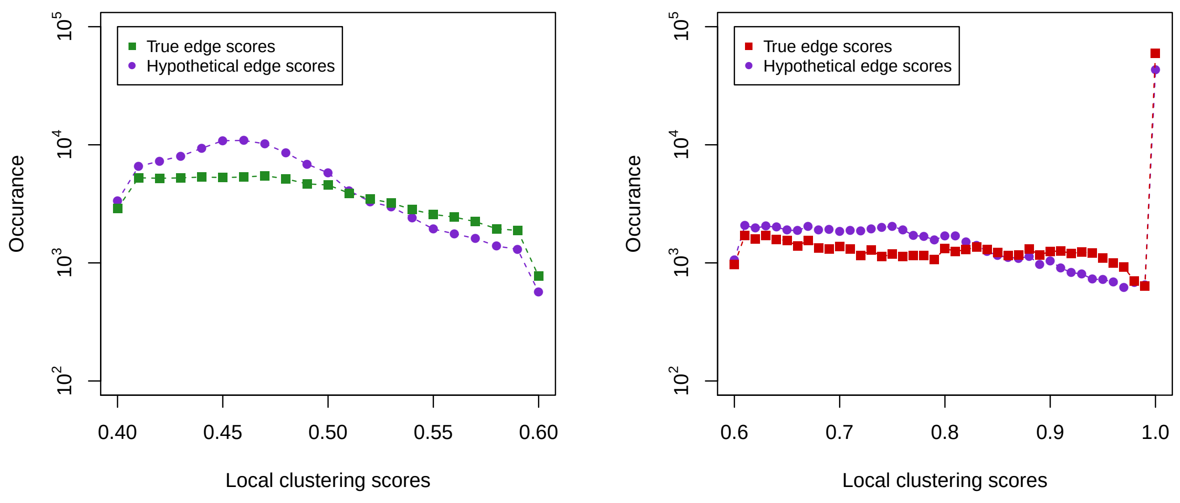
Table 4.13: Neighbor pairs for YFL044C and YNL155W. The pair under investigation is highlighted blue for YFL044C and red for YNL155W. The entries for YNL155W explain the high score of 1.0. None of its neighbor pairs are connected with each other directly excpet the one highlighted red. This pair is connected indirectly through YDL190C (Fig. 4.7).

The normalization of the scores for each ego-vertex causes the main fluctuation on the thresholds. The scores for the neighbor pairs correspond only to the ego-vertex. As soon as we change the ego-vertex we have to change our definition for the thresholds with respect to it. A neighbor pair can get a very low score in one egocentric network if some other pairs in the same network are assigned a very high score. The same pair can exhibit relatively stronger ties in other egocentric networks in the vicinity. The main challenge is to find out a suitable measure out of the different scores for its tie strength. We have taken the average value while the median can be also taken into account. If a neighbor pair exhibits a significant tie strength in all the egocentric networks it belongs to we can assume a putative *friendship* between them.

4.4.3 Suggested neighbor pairs in the EXCERBT network

The MapReduce program for the *friend finder* algorithm had a runtime of exactly 2 hour in case of the EXCERBT network. It had 341,854 ego-vertex assignments. We could have extended our restriction of a highest degree of 5,000 very well. The output data was around 180 GB in size which is in fact enormous comparing to the input data size of 1.5 GB. But these output data can be processed further and divided into two different files according to the tie strength.

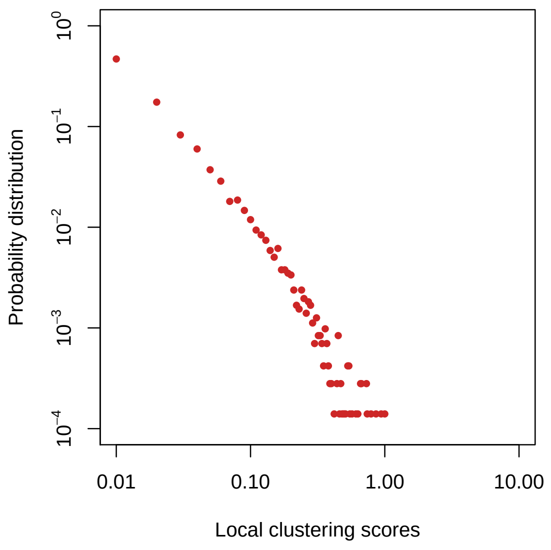
We assigned 0.6 as the threshold for strong ties and 0.4 for the weak ones arbitrarily. We had our third reducer (see 4.3.2) produce two different outputs for the two different tie strengths. As already mentioned in section 4.3.3 we registered the potential connections only, i.e. pairs with an edge tag of "0". Due to their membership in multiple egocentric networks they had multiple scores accordingly. The average score was computed by the additional last reducer which also had two outputs like its forerunner. The average acores did not fluctuate much from the original normalized scores. Because the third reducer already extracted and sorted the pairs according to the two respective thresholds. Finally the last reducer delivered a list of the frequency of hypothetical edges for a particular average score.



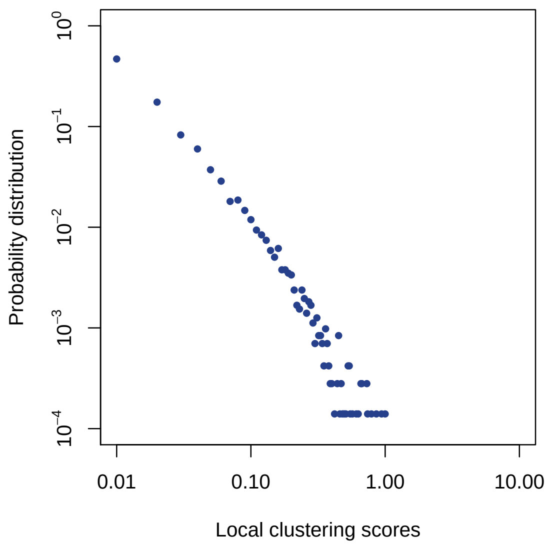
(a) Score distribution of the edges within the range of threshold 1, i.e. between 0.4 and 0.6. 108,984 hypothetical edges and 79,647 true edges were found within this threshold.

(b) Score distribution of the edges within the range of threshold 2, i.e. between 0.6 and 1.0. 100,951 hypothetical edges and 109,252 true edges were found within this threshold.

Figure 4.8: Comparison of the local clustering score distribution of tie scores of both true and hypothetical edges of the EXCERBT network. Note that the y-axis which represents the occurance here is a log axis.



(a) Log-log plot of the tie scores distribution of *parkinson disease* extracted from the EXCERBT network.



(b) Log-log plot of the tie scores of *parkinson disease* from the Parkinson's disease gene network.

Figure 4.9: Comparison of the probability distribution of tie scores of *parkinson disease* in two networks. The scores for the local clustering are plotted on the x-axis and the y-axis states the probability of the scores.

We plotted this frequency for the two different threshold ranges in Fig. 4.8. The noticable high altitude for score 1.0 in Fig. 4.8b is quite obvious. This score represents the neighbor pair with the highest score for one egocentric network and there were 341,854 of them in this case.

We also extracted the ego-vertex *parkinson disease* from the output of the EXCERBT network and plotted the scores in a log-log plot (Fig. 4.9a). After that we plotted the scores for the same entity (*parkinson disease*) from the Parkinson's disease gene network (Fig. 4.9b) and compared the scores. No discrepancy could be observed between the two plots so far. In both cases we obtained exactly 9,045 suggestions for possible edges.

4.4.4 Neighbor pairs for *parkinson disease*

The execution on our Parkinson's disease gene network took only 1 minute. The gene NF-kappaB (as entity *nfkappab*) had the highest degree of 2,690. As mentioned in section 4.4.1 we determined the two thresholds for the entity *parkinson disease* from the distribution of the scores computed for it. A connection score from 0.0 to 0.1 can be regarded as very weak and there are 6,538 neighbor pairs with such weak connections. For this reason we set the threshold for weak ties at 0.2 experimentally. On the other hand we found only 13 neighbor pairs with a tie score ranging from 0.6 to 1.0. The number of neighbor pairs increased rapidly below the score of 0.6. Therefore we chose 0.6 as the threshold for strong ties (Fig. 4.10).

We found 191 neighbor pairs with weak tie between them. 156 of them have direct connections and 35 have indirect connections with each other. As for strong tie, all of the 13

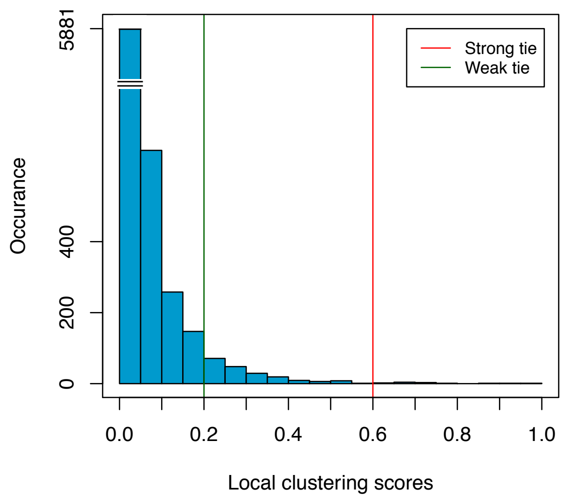


Figure 4.10: Determining a decent threshold for the neighbor pairs of the entity *parkinson disease*. The green margine identifies the threshold for weak ties and the red one for strong ties.

suggested neighbor pairs are connected directly. We visualized the existing connections, both weak and strong, for *parkinson disease* in Fig. 4.11. We investigated further and selected the gene ERK1/2 (as entity *erk12*) for the illustration of all of its connections above the two thresholds. No strong ties with indirect connection could be detected here.

ERK1/2: The term ERK1/2 stands actually for ERK1 and ERK2. ERK ist the abbreviated form of *extracellular signal-regulated kinase*. ERK1 and ERK2 are also known as *mitogen-activated protein kinase 3* or for short MAPK3 and MAPK1 respectively. Proteins encoded by these genes are members of the MAP Kinase family. According to the EXCERBT database it has 872 direct neighbors including *parkinson disease*. An example of the connections found between ERK1/2 and *parkinson disease* in the EXCERBT database is as follows[3]:

"Degenerating neurons of Parkinson's disease (PD) patient brains exhibit granules of phosphorylated extracellular signal-regulated protein kinase 1/2 (ERK1/2) that localize to autophagocytosed mitochondria."[32]

Fig. 4.12 presents a graphical visualization of the strong and weak ties suggested between *erk12* and its tuple partners. In the following we have briefly discussed 4 pairs with ERK1/2.

ERK1/2 and NF-kappaB

The gene NF-kappaB (*nuclear factor kappa-light-chain-enhancer of activated B cells*) is a transcription regulator. It plays a key role in the regulation of immune response to infection. The entity *nfkappab* has a strong and direct connetion to *erk12* (Fig. 4.11). They share 834 common neighbors which is a high number comparing to 872 direct neighbors of ERK1/2. As such the strong tie between ERK1/2 and NF-kappaB is justified. As for direct connection, we found out four distinct relation types between them in the EXCERBT database with ERK1/2 as source and NF-kappaB as target:

[3]`http://mips.helmholtz-muenchen.de/excerbt/#1347487764268`, September 26, 2012

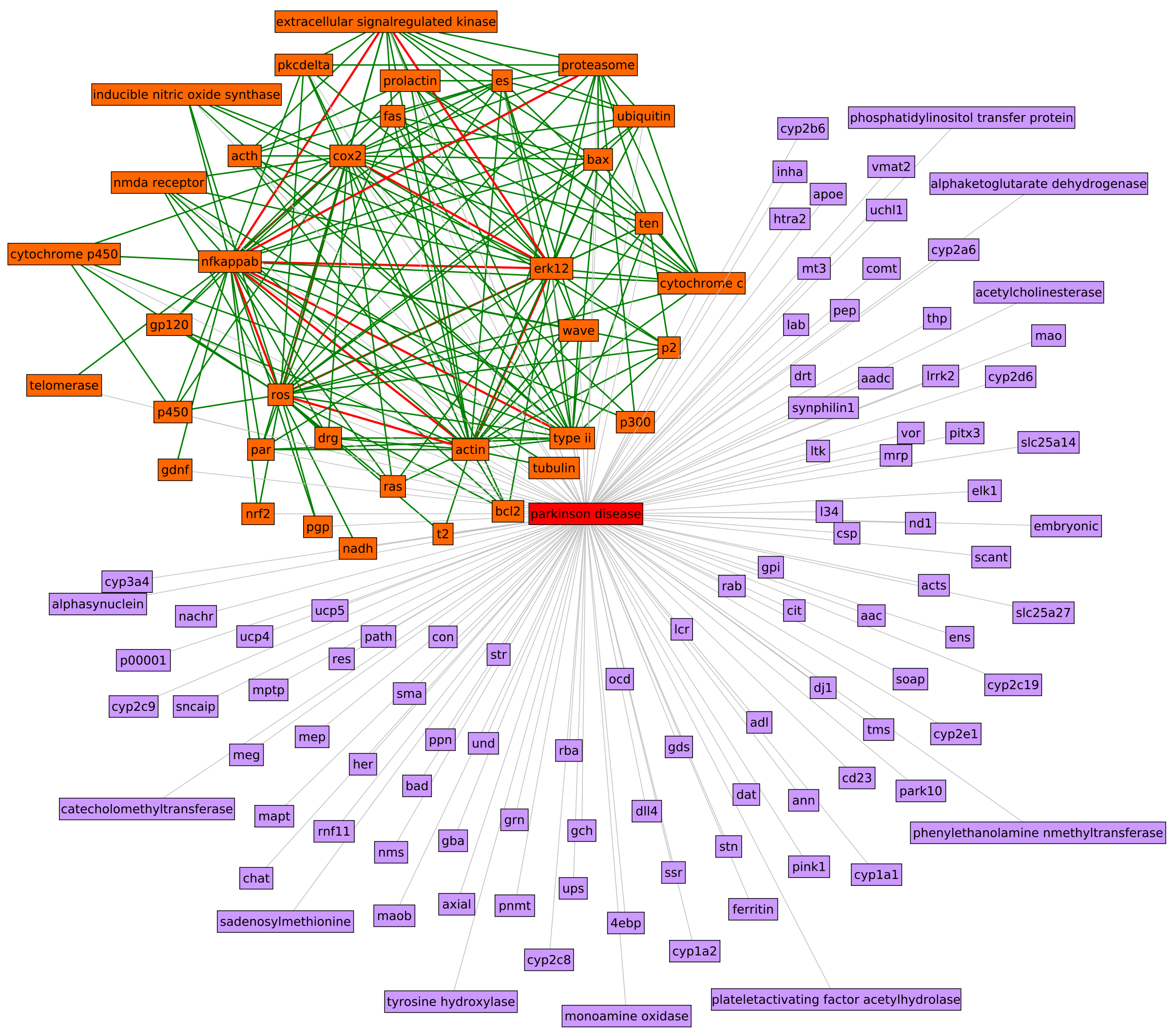


Figure 4.11: Graphical visualization of suggested neighbor groups that already exist for *parkinson disease*. The solid green lines stand for weak ties and the red ones for strong ties between existing edges in each case. The light gray edges indicate the direct connections of the entities to the ego-vertex *parkinson disease*.

1. at_regulation_act
2. at_regulation_pas
3. at_activation_act
4. at_inhibition_pas

Semantically ERK1/2 is involved in activation of NF-kappaB and NF-kappaB is involved in inhibition of ERK1/2. Both are summarized as active and passive regulation between each other.

ERK1/2 and cytochrome c

Cytochrome c is a small heme protein that functions as a central component of the electron transport chain in mitochondria. It is also involved in the process of initating apoptosis. Numerous processed pseudogenes of this gene are found throughout the human genome[4].

[4]`http://www.ncbi.nlm.nih.gov/gene/54205`, September 26, 2012

The gene that encodes this protein is CYCS and is entered as a synonym to cytochrome c in the EXCERBT synonym database. Hence the query for only genes as target type for our Parkinson's disease gene network included cytochrome c. It shares 313 common neighbors with ERK1/2 which explains the weak tie with respect to NF-kappaB.

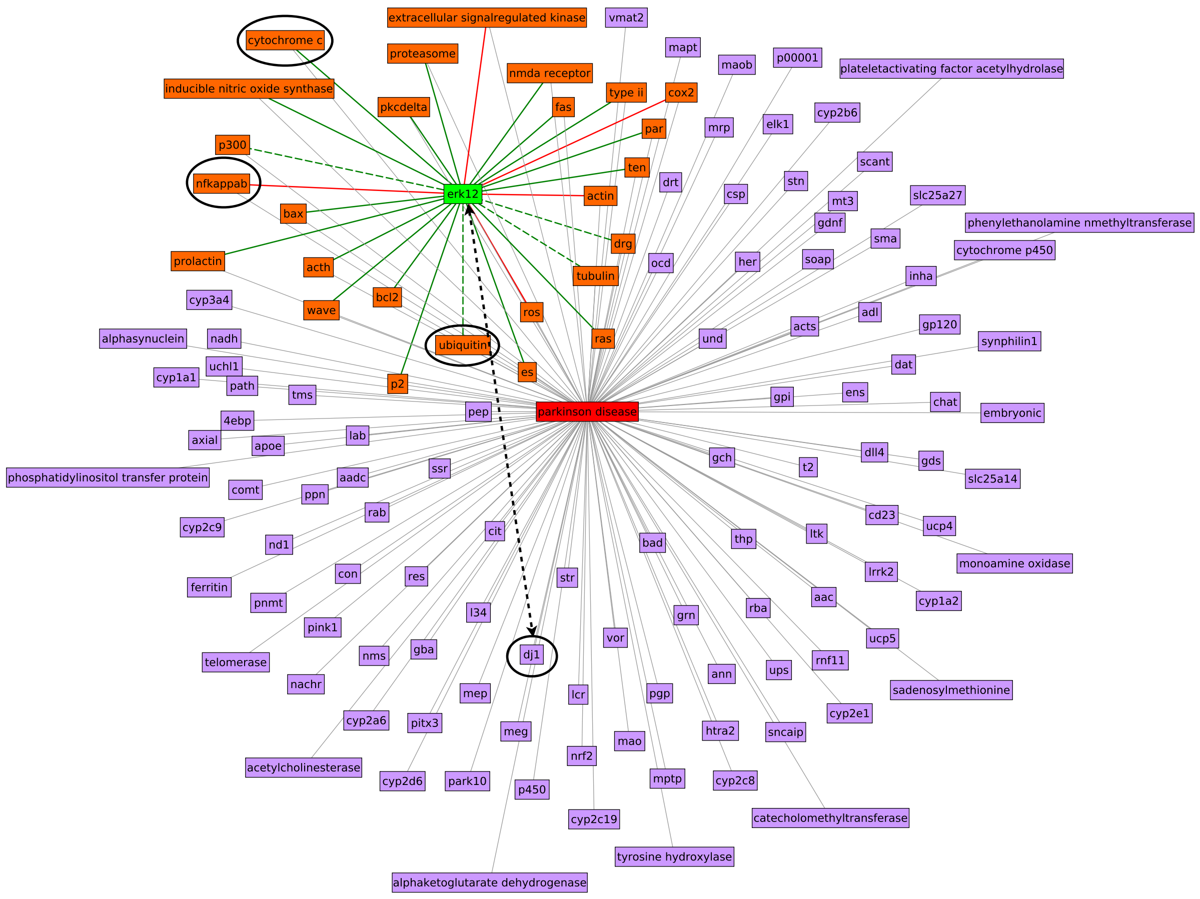


Figure 4.12: Graphical visualization of existing and non existing edges between neighbor pairs of *parkinson disease* containing *erk12*. The green lines stand for weak ties and the red ones for strong ties. On an addition the solid lines show the existing edges whereas the dashed ones the non existing ones. The light gray edges indicate the direct connections of the entities to the ego-vertex *parkinson disease*. The black dashed line is to point out the entity *dj1* with which *erk12* has no significant tie. We have taken the entities in the ellipses as examles to investigate.

We found 2 direct relations between them, namely:

1. regulation_act
2. inhibition_act

Semantically ERK1/2 is related to the process of cytochrome c inhibition and thus its regulation too.

ERK1/2 and ubiquitin

Ubiquitin is a small regulatory protein found ubiquitously in almost all tissues of eukaryotic organisms. In mammals it is encoded by 4 different genes. The process of marking a protein with ubiquitin is called ubiquitination. It is known to play central roles in the regulation of various cellular processes such as protein degradation, protein trafficking, cell cycle regulation, DNA repair, apoptosis and signal tranduction [33]. The query for target tape gene hit ubiquitin probably because the term *ubiquitin* was registered semantically as the super group for ubiquitin genes. We could found not a single direct reletion between ERK1/2 and ubiqutin in the EXCERBT database (hence the dashed green line in Fig. 4.12). But they have 304 common neighbors which indicates a weak tie. We went deeper and found NF-kappaB as a common neighbor. Ubiquitin is related to NF-kappaB regarding activation, inhibiton and regulation. So if both ubiquitin and ERK1/2 are involved in the regulatory process of NF-kappaB there must be some sort of relation between them.

ERK1/2 and DJ-1

DJ-1 is also known as the PARK7 gene which encodes the PARK7 protein in humans. Defects in this gene result into the autosomal recessive early-onset arkinson disease 7[5]. We found two direct relations between ERK1/2 and DJ-1 which state that the latter is involved in activation and regulation of the former. They share only 62 neighbors with each other. Consequently the tie strength between them was beyond any of the two thresholds used in this work.

This was simply a novel attempt to cluster a massive-scale network locally with an egocentric concept. Obviously there are scopes to improve it. The first and foremost challenge in this work was defining legitimate thresholds which still remains. In all attempts related to the egocentric local clustering we selected some arbitrary scores as thresholds. Then we refined our definition of thresholds empirically. We recommend a further inspection between the scores alloted to the vertices contained in a neighbor pair and their clustering coefficients.

Another aspect to explore could be to change the type of vertices chosen as neighbors' neighbors. For example, one chooses metabolites as target type for the genes which are directly connected to a phenotype say *parkinson disease*.

The hypothetical edges found in the egocentric network of Parkinson's disease gene should be examined from other egocentric networks' point of view. By means of the methods presented in this chapter we found out numerous hypothetical edges above the tie strength thresholds (Fig. 4.8). Neighbor pairs with such hypothetical edges should be investigated further. It points out the fact that an evidence of publication regarding them is not available yet (see 4.4.4). Pairs with a strong tie ought be connected in reality since they belong to the same cluster. They share the same information and must have publications accordingly. If not then it is essential to start researching about them. This realization can open a broad horizon for scientists and researchers.

[5] `http://www.ncbi.nlm.nih.gov/gene/11315`, September 26, 2012

5 Biomedical Synonym Resolution

According to the definition of the Oxford dictionary a synonym is a word or phrase that means exactly or nearly the same as another word or phrase in the same language. The state of being a synonym is defined as synonymy. In the context of biomedical literatures it is one of the most important relations found between different terminologies [34]. The language of biomedical texts like all natural languages has a complex structure and creates the synonym problem. It is crucially important to take synonymy into account in order to build high quality text mining systems for biomedical literatures. In this chapter we offer a proposition to resolve the synonym problem by applying our egocentric local clustering method introduced in the previous chapter. This attempt does not involve any text mining, it clusters rather the synonymous terms resulting from the text mining and groups them together.

5.1 The EXCERBT network: a biomedical thesaurus

The number of publications that were analyzed by the tool EXCERBT lies around 21.8 millions. The synonymous terms found among them are listed in an enormous synonym database. Alone 544 synonyms can be found for *aspirin*[1]. Unfortunately these synonyms are not registered under one key name or ID and hence one can find 127 synonyms for the term *Aspirin*[2]. These two sets of synonyms are also very redundant and the redundancy continues for each of the elements in one set[3]. We experimented with the Parkinson's disease gene network as a strategy approach towards the solution. Our basic concept was to build up tuples of synonyms and observe the local clustering scores assigned to them by means of the egocentric local clustering algorithm (4). One cannot ignore special characters or spelling of a term when it comes to synonyms. Therefore we used the raw version of the Parkinson's disease gene network that we extracted from the EXCERBT network for this experiment (see 2.1.2).

5.2 Method

The following steps were performed for the experiment to detect synonyms:

1. We first clustered the original Parkinson's disease gene network with the help of the program for egocentric local clustering.

2. Next we determined the thresholds for strong ties and weak ties from the local clustering scores (Fig. 5.1).

[1]`http://mips.helmholtz-muenchen.de/excerbt/rest/entities/synonyms/aspirin`, September 26, 2012

[2]`http://mips.helmholtz-muenchen.de/excerbt/rest/entities/synonyms/Aspirin`, September 26, 2012

[3]`http://mips.helmholtz-muenchen.de/excerbt/rest/entities/synonyms/Toldex`, September 26, 2012

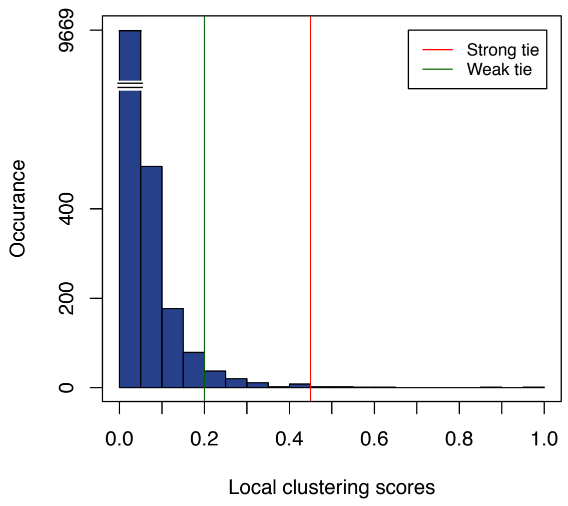


Figure 5.1: Defining the threshold for the neighbor pairs of the entity *parkinson disease*. The green margine identifies the threshold for weak ties and the red one for strong ties.

3. Then we scanned all of the suggested neighbor pairs for synonymy. We set the case "synonymy is true" for a pair $\langle A, B \rangle$ if at least one of the following applies:
   a) **Symmetry**: If $A \equiv B$. Automatically $B \equiv A$.
   b) **Transitivity**: In case symmetry does not apply, then for an element $x \in$ synonym set of A, i.e. $x \equiv A$, if $x \equiv B$ then $A \equiv B$.
4. We observed the behavior of the scores of the synonym pairs found against that of all other neighbor pairs, followed by grouping of the synonyms. We grouped the synonyms so that we could perform further tests with one group.
5. After that we executed our clustering program once again on the raw Parkinson's disease gene network. But this time we considered only members of one synonym group and their neighbors for the egocentric network of *parkinson disease*.
6. Subsequently we plotted the local clustering scores of the synonyms once again and interpreted the tie strength between a pair.

5.3 Result

The procedure listed in the previous section delivered the following result:

1. The clustering of the *de facto* Parkinson's disease gene network showed that the entities *parkinson disease* and *Parkinson disease* had exactly the same 142 neighbors and consequently exactly the same 10,011 pairs of neighbors were suggested. So we carried on with only one entity as the ego vertex, namely *parkinson disease*.
2. After the clustering we plotted the local clustering scores in a histogram (Fig. 5.1). We took 0.2 as the threshold for weak ties as in the previous chapter (see 4.4.4) but tuned the threshold for strong ties down to 0.4. We found 16 pairs with strong ties which also exist in reality. No hypothetical strong edges were found.

3. We found only 68 synonym pairs out of the 10,011 possible edges. 5 pairs were no true synonyms rather simply the same word with slightly different spelling (Tab. 5.1).

Neighbor 1	Neighbor 2
P300	p300
4E-BP	4EBP
DJ-1	DJ1
UCH-L1	UCHL1
COX-2	COX2

Table 5.1: The five *non true* synonym pairs found in the original Parkinson's disease gene network.

4. The scores for the other 63 true synonym pairs were extremely low ranging from 0.01 to 0.07 (Fig. 5.2). We detected two groups of synonyms here. One group contained the entities *ACTH*, *PPN*, *MPTP* and *MEG* with 13 pairs. The other group contained genes related to the *cytochrome p450* super family with 50 pairs. This group consisted of 11 genes. So theoretically there should be $\binom{11}{2} = 55$ pairs. Since our scanning program searched for synonyms only it found 50 pairs containing synonymous terms. It is quite possible that the two contributing genes in the other 5 putative pairs were not registered as synonyms in the EXCERBT synonym database.

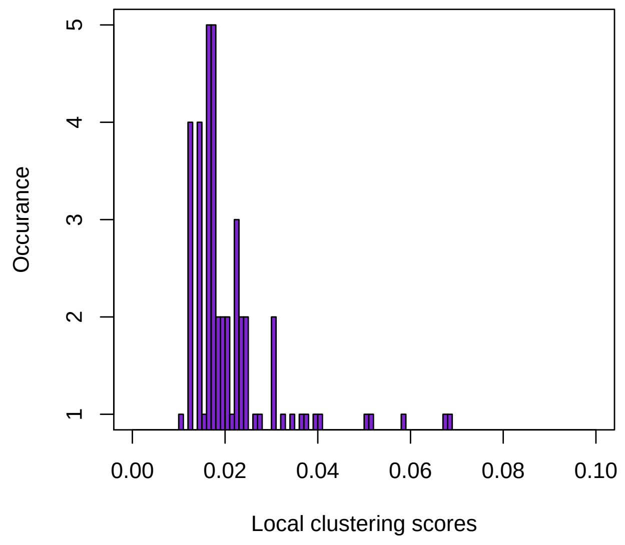


Figure 5.2: Illustration of the local clustering scores resulting for synonyms after the clustering of the raw version of Parkinson's disease gene network. We have taken the range for the x-axis from 0.00 to 0.10 instead of 1.0 since the scores of the 63 true synonym pairs range from 0.01 to 0.07.

5. We chose to continue the experiment with the synonyms of the *cytochrome p450* super family. We eliminated all other vertices and their neighbors except the synonymous vertices and their neighbors from the original Parkinson's disease gene network.

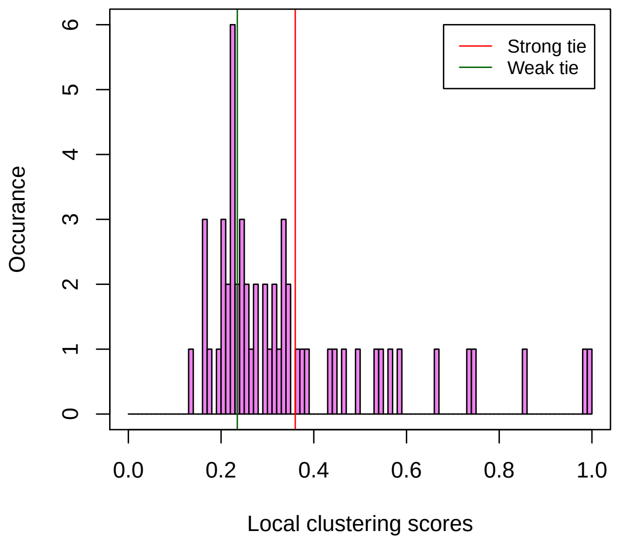


Figure 5.3: Determining the threshold for neighbor pairs of synonymous vertices of the gene *cytochrome p450* super family. The green margine identifies the threshold for weak ties and the red one for strong ties.

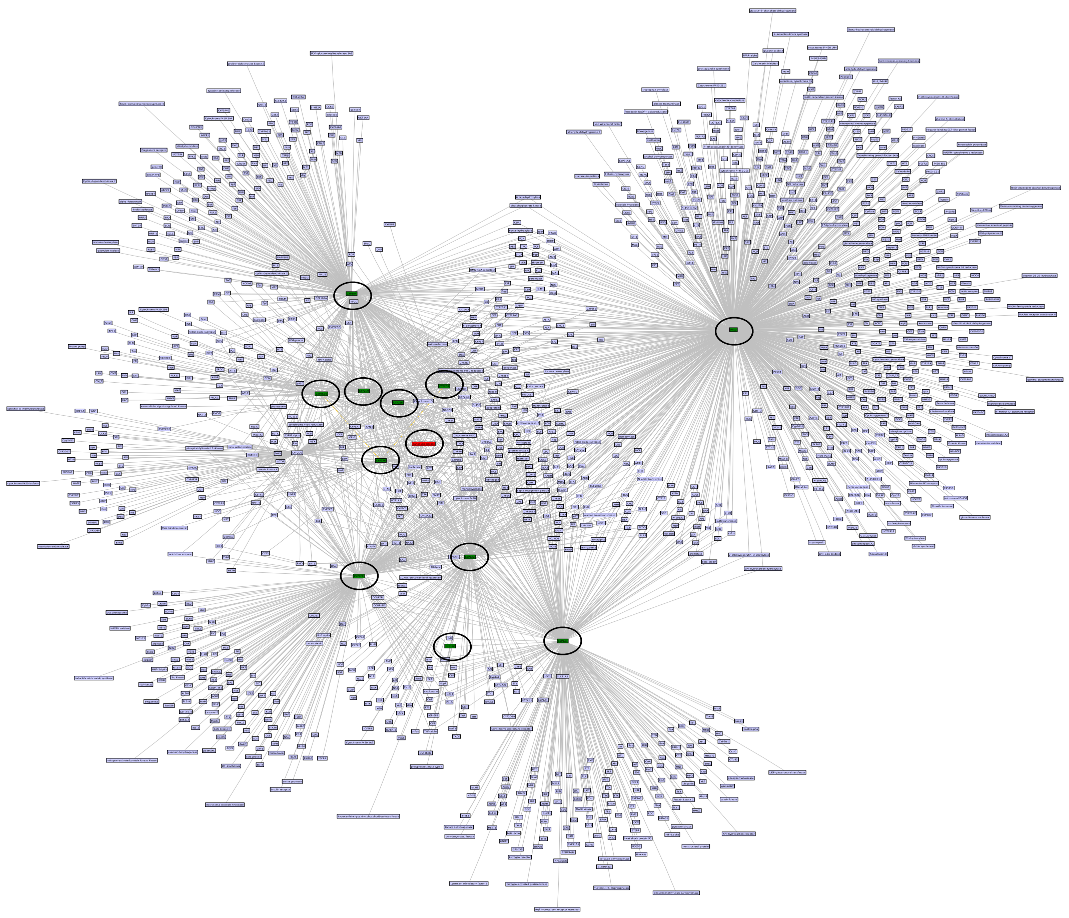


Figure 5.4: Graphical visualization of the synonym clusters of *cytochrome p450*. The red vertex in the middle is the entity *parkinson disease* and the dark green vertices are the 11 synonymous genes of *cytochrome p450*.

6. Finally we plotted the local clustering scores of the synonym pairs as a histogram (Fig. 5.3). We selected 0.23 as the threshold for weak ties and 0.36 for strong ties. We visualized these pairs as a subnetwork of the original Parkinson's disease gene network in Figs. 5.4 and 5.5.

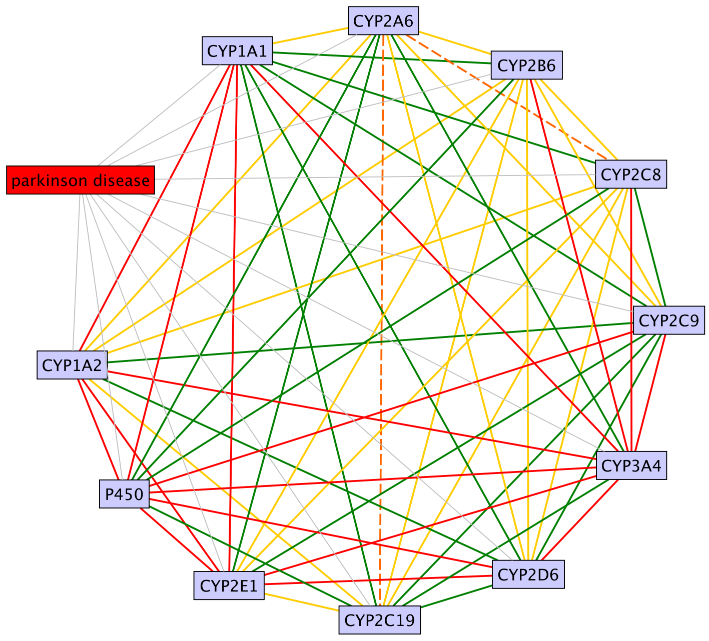


Figure 5.5: Graphical visualization of the pairs containing synonymous genes of *cytochrome p450*. The red lines represent strong ties, the green ones weak ties and the yellow ones all other ties beyond the thresholds. The only two orange dashed lines show the two pairs which are not connected directly. Their scores are also below both thresholds.

5.4 Discussion

The intention of eliminating the nonsynonymous vertices was to exclude the influence of neighbor pairs with higher number of indirect connections from the egocentric network. Say the vertices A and B have 100 and 200 direct neighbors respectively but share 50 common neighbors. On the other hand the synonym pairs X and Y in the same egocentric network have 10 direct neighbors each and share all of them. The egocentric local clustering algorithm would assign the former a higher score than the latter and thus the score normalization would deliver a minute score for the synonym pair. Since we were trying to find out possible synonym candidates we had to exclude the nonsynonym ones for this experiment.

Out of the 55 possible neighbor tuples 17 were detected to exhibit strong ties and 21 weak ties. The rest had a score beyond the two thresholds. Two of them were not connected directly. So we might assume the two contributing vertices of any member tuple showing a strong tie to be synonyms. Considering the fact that the synonym tuples were under investigation from the point of view of only one egocentric network the result is neither informative nor convincing. At this point the procedure needs strategic and technical refinement.

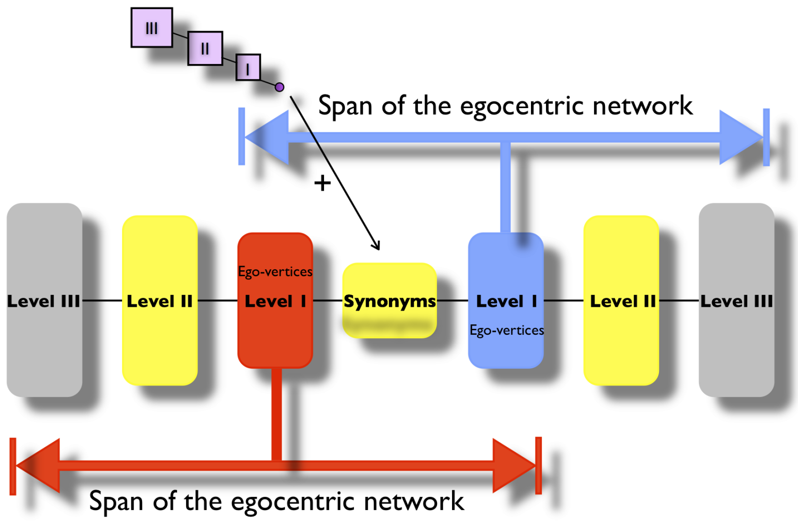


Figure 5.6: Proposition how to choose the vertices for the network with the synonyms. The *level I* vertices (red and blue sets) should form the egocentric networks. The set of synonym vertices would be then within the respective ranges (red and blue arrows) for the set of egocentric networks. After determining the scores for the synonym pairs a vertex (purple), of which the synonymy is unknown, is to be added along with its neighbor sets up to depth 3 to the test set.

5.5 Proposition

Under the circumstances we propose the following two improvements regarding the experiment to tackle the challenge of finding synonyms:

Strategic approach

We have observed the synonym pairs within one egocentric network. We need to investigate the pairs from other egocentric networks in the vicinity as well. To be able to do that we need to revise the formation of the main network. One takes the synonymous terms and includes all vertices within the range of depth 3 into the network. The vertices at level 1 are to be marked as the ego-vertices which form the set of egocentric networks (Fig. 5.6). This strategy would allow the membership of the synonym pairs in multiple egocentric networks. Other pairs from level 2 (yellow sets in Fig. 5.6) which do not include any synonyms and cross pairs of synonyms and non synonyms are ignored. Then the scores for one synonym pair assigned by the multiple ego-vertices should be averaged. Defining a threshold at this point does not make that much of sense since we know that all of these pairs are synonyms beforehand. But the score range is important.

Technical approach

Another refinement would be to develop a more accurate scoring function. Until now we have assigned "1" for a direct connection and "0.25" for an indirect connection between pairs. As already explained in the previous section (see 5.4) this scoring approach for indirect connections turns out to be naive if it comes to the synonyms. A new concept would be to allow scores depending on the percentage of common neighbors between the contributing vertices in a pair. One sums up the numbers of direct neighbors of each vertex in a pair, then counts the number of their common neighbors and finally determines the percentage of the common neighbor out of all neighbors. A score of "0.1" should be alotted for each percent. Let $\langle a, b \rangle$ be a synonym pair and A and B be their respective neighbor sets. Then the score S for the indirect connections between $\langle a, b \rangle$ would be the following:

$$S = \frac{|A \cap B|}{|A \cup B|} \times 100 \times 0.1 \tag{5.1}$$

If $a \in B$ or $b \in A$ then there is a direct connection and the score should be "1" in this case. In this way if the pair shares 100% of their neighbors they would get a score of "10" and for a direct connection "1".

After all these procedures are accomplished one adds a new vertex with its neighbors up to depth 3 in this network. The vertex itself is added to the set of synonym vertices. The clustering is performed again. The new pairs with the newly added are examined and the scores are compared to the original scores of the synonym pairs determined previously. Only after that one can postulate whether the new vertex could be a synonym or not.

It should be clear to the reader that the egocentric local clustering algorithm would detect possible synonym candidates and cluster them with the existing synonyms in the EXCERBT synonym database. This algorithm does not guarantee a 100% discovery of synonyms. Even if we consider the inclusion of a putative synonym candidate upon a strong tie it is still possible that a synonym candidate without any synonymy might fulfill the requirement by chance. Yet it can be apprehended that this egocentric local clustering would detect most of the synonyms and cluster them with the respective groups. Therefore it is not farfetched at all to experiment with our proposition introduced above.

6 Conclusion & Outlook

The number of biomedical articles is growing exponentially and is yet to grow in time to come. It is and will remain too large a consortium for an individual to read and interpret all in a practicable time. Automated text mining systems are hence a must for the conversion of this amount of information into knowledge which also makes the implementation of networks inevitable. Today we are almost overstrained in dealing with data at terabyte and petabyte scales. Due to the exponential growth of biomedical literature that day is not so far when we will be compelled to tackle data at zettabyte (10^{21}) or even yottabyte (10^{24}) level.

In this work we have integrated MapReduce framework to cope up with this massiveness. But even MapReduce is no panacea if handled carelessly [11]. Designing the algorithms remains crucial to success of the algorithms. Space requirements of algorithms do not disappear simply because there are more machines available. For a large enough n, say $n \approx 10^8$, an $O(n^2)$ memory is still an unreasonable request which would require approximately 10 petabytes of storage. Therefore the programmer has to mind the facts about machine memory, total memory and number of each MapReduce round. Almost in the end of our work we figured out the implementation of an even better designing for the MapReduce algorithm to compute clustering coefficients (2). This design would decrease the runtime and storage problem to a large extent. Since it would go beyond the span of a bachelor thesis we did not carry the design into effect. But we would strongly recommend an implementation of the new design.

We detected an unambiguous scale-freeness in the EXCERBT network proving once again that most biological networks are such, even when it comes to networks derived from biomedical publications. Concerning the EXCERBT network itself one can also consider the assignments of direction and weight into it. The refinement of the algorithm design would make it possible to include complexities like directed and weighted edges. The different relation types between entities in the EXCERBT network can be considered as unit for weight while the specifications *"_act"* and *"_pas"* can be applied as directions. Say, two entities show the following types of relations between them:

1. activation_act
2. inhibition_pas
3. regulation_act
4. regulation_pas
5. expression_act

The first four terms should be added up together since *activation* and *inhibition* are comprised in *regulation* and there ought to be a bidirectional relation assigned to the entities. Furthermore there should be a weight of 2 units between them due to the extra factor *expression*.

As for the egocentric local clustering algorithm there is also room for improvement regarding runtime, storage and above all the designing. We have already established the basic concept for a search engine to find possible new relations between entities. We discovered quite a few of such relations and presented the example of the weak tie found between ERK1/2 and ubiquitin (see 4.4.4). We have taken only related entities of type *gene* for Parkinson's disease. It was necessary since one cannot step directly into a complicated level while experimenting with a novel attempt. Creating multiple catagories in the EXCERBT network according to the entity types can be the next criteria for the clustering. The concept is to create parallel networks within the original network. The entities in the EXCERBT network exhibit multiple natures referring to their types like *gene*, *protein*, *metabolite* etc. As for example the entity ERK1 is connected to ARIA which can be reffered to as *gene* and *metabolite* alike. In this work we assigned scores to a neighbor tuple in an egocentric network counting simply the number of connections between them, i.e. if the tuple $\langle A, B \rangle$ has a common neighbor C then we assigned a score of 0.25 to the tuple (see 4.3.2). Say both A and B belong to the gene and the metabolite networks. If C belongs to only one of the two parallel networks then the tuple $\langle A, B \rangle$ would gain a higher score than the case if C belonged to only one of the two parallel networks. Such quantification can lead to a better qualification of the tie strength. More concretely, adding this aspect would find out putative connections based on the contribution of the dispersed edges in the multiple networks. But the number of the multiple networks must be limited in order to avoid a possible exponentiality.

The other demanding prospect was the score calculation and the threshold determination. We believe that the proposition regarding the score calculation mentioned in previous chapter offers an efficient solution to the problem (see 5.5). Regarding the synonyms we detected a tendency amongst them of being in the same cluster of the superfamily or group. Hence we suggested that a synonym candidate would also tend to belong to the respective cluster upon a true synonymy. There are at least half a dozen of text mining approaches to detect synonyms alone. Identifying synonyms after the automation is not at all mundane. This experiment is still at its primitive level but we expect an affirmative result upon the accomplishment of our proposition.

Summing up the whole thesis, the attempt to analyze the EXCERBT network was successful. We could caculated the degrees for all of the entities with the help of HiveQL and the clustering coefficients for entities with a degree of $\leq$10,000 by our MapReduce algorithm. The local clustering attempt was also successful though it did not go smoothly all along. Nevertheless we could cluster entities locally with a degree of $\leq$5,000 by creating 341,854 egocentric networks. We detected 108,984 and 100,951 hypothetical edges with weak and strong ties respectively. In case of Parkinson's disease we could identify 191 edges with weak tie and 13 with strong tie. All of the edges with strong tie exist in reality. 35 out of the 191 edges with weak tie were hypothetical and need to be investigated further. Although we clustered the synonyms within the Parkinson's disease network, we could not achieve any effective or convincing result to distinguish synonyms perfectly. But this led us into a promising proposition for the synonym resolution in turn.

Bioinformatics makes extensive use of information theoretic measures as part of the attempt to use expert computational data analysis tools on biological problems. Bioinformaticians are trained to have the skill and ability to identify patterns and correlations among the data. With this explosion of data has come the opportunity to develop an evermore enigmatic understanding of the way organisms function at their most fundamental levels. Genes are being identified that are involved in basic human experiences. Genes and gene regulatory mechanisms are being rapidly identified that underlie many of the most dreaded diseases including cancer, heart disease and the degenerative neurological diseases of aging [35]. All of this information can easily sway our focus from the fields that have not been researched yet. Our entire work was based upon a network derived from biomedical publications. Upon a flawless implementation of our local clustering algorithm we provide the researcher and scientist community with a tool to enable new discoveries.

Acknowledgements

My gratitude to Prof. Dr. Hans-Werner Mewes for providing me this thesis work and to Dr. Volker Stuempflen for offering me the topic. Many many thanks to my supervisor Benedikt Wachinger for all the input and ideas, discussions, advices and foremost his patience. A very special thank goes to Robert Strache for his useful and critical suggestions which motivated me a lot and not to mention his excellent coffee. Thanks to Malte Kessner for the proof reading. I am also grateful to my family for their support throughout my work. Last but not least I would like to thank the whole *Biological Information System* group of Helmholtz Zentrum Munich and my best friend Julia Krumhoff for a most pleasant working atmosphere during the summer spent inside.

7 Appendix

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Andrey Rzhetsky, Michael Seringhaus, and Mark B. Gerstein. Getting started in text mining: part two. *PLoS Comput Biol*, 5(7):e1000411, Jul 2009.

[2] K Bretonnel Cohen and Lawrence Hunter. Getting started in text mining. *PLoS Comput Biol*, 4(1):e20, Jan 2008.

[3] Anna Stavrianou, Periklis Andritsos, and Nicolas Nicoloyannis. Overview and semantic issues of text mining. *SIGMOD Rec.*, 36(3):23–34, September 2007.

[4] Masashi Tanaka, Takeshi Takeyasu, Noriyuki Fuku, Guo Li-Jun, and Miyuki Kurata. Mitochondrial genome single nucleotide polymorphisms and their phenotypes in the japanese. *Ann N Y Acad Sci*, 1011:7–20, Apr 2004.

[5] Clement A. Gautier, Emilie Giaime, Erica Caballero, Lucía Núñez, Zhiyin Song, David Chan, Carlos Villalobos, and Jie Shen. Regulation of mitochondrial permeability transition pore by pink1. *Mol Neurodegener*, 7:22, 2012.

[6] Hans-Werner Mewes, Andreas Ruepp, Fabian Theis, Thomas Rattei, Mathias Walter, Dmitrij Frishman, Karsten Suhre, Manuel Spannagl, Klaus F X. Mayer, Volker Stümpflen, and Alexey Antonov. Mips: curated databases and comprehensive secondary data resources in 2010. *Nucleic Acids Res*, 39(Database issue):D220–D224, Jan 2011.

[7] Thorsten Barnickel, Jason Weston, Ronan Collobert, Hans-Werner Mewes, and Volker Stümpflen. Large scale application of neural network based semantic role labeling for automated relation extraction from biomedical texts. *PLoS One*, 4(7):e6393, 2009.

[8] Haiyuan Yu, Pascal Braun, Muhammed A. Yildirim, Irma Lemmens, Kavitha Venkatesan, Julie Sahalie, Tomoko Hirozane-Kishikawa, Fana Gebreab, Na Li, Nicolas Simonis, Tong Hao, Jean-François Rual, Amélie Dricot, Alexei Vazquez, Ryan R. Murray, Christophe Simon, Leah Tardivo, Stanley Tam, Nenad Svrzikapa, Changyu Fan, Anne-Sophie de Smet, Adriana Motyl, Michael E. Hudson, Juyong Park, Xiaofeng Xin, Michael E. Cusick, Troy Moore, Charlie Boone, Michael Snyder, Frederick P. Roth, Albert-László Barabási, Jan Tavernier, David E. Hill, and Marc Vidal. High-quality binary protein interaction map of the yeast interactome network. *Science*, 322(5898):104–110, Oct 2008.

[9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *SOSP*, pages 29–43, 2003.

[10] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system.

[11] Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, pages 607–614, 2011.

[12] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[13] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive - a warehousing solution over a map-reduce framework. *PVLDB*, 2(2):1626–1629, 2009.

[14] Joshua O'Madadhain, Danyel Fisher, Scott White, Padhraic Smyth, and Yan biao Boey. Analysis and visualization of network data using jung.

[15] Reinhard Diestel. *Graphentheorie (3. Aufl.).* Springer, 2006.

[16] Albert-László Barabási and Zoltán N. Oltvai. Network biology: understanding the cell's functional organization. *Nat Rev Genet*, 5(2):101–113, Feb 2004.

[17] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, Jun 1998.

[18] P. Erdős and A. Rényi. On the evolution of random graphs, 1960.

[19] M E J. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. *Proc Natl Acad Sci U S A*, 99 Suppl 1:2566–2572, Feb 2002.

[20] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, Oct 1999.

[21] Reuven Cohen and Shlomo Havlin. Scale-free networks are ultrasmall. *Phys Rev Lett*, 90(5):058701, Feb 2003.

[22] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A. L. Barabási. Hierarchical organization of modularity in metabolic networks. *Science*, 297(5586):1551–1555, Aug 2002.

[23] Jae Dong Noh. Exact scaling properties of a hierarchical network model. *Phys. Rev. E*, 67,:R045103., 2003.

[24] N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory, 1736-1936.* Clarendon Press, New York, NY, USA, 1986.

[25] John P. Scott. Social network analysis: developments, advances, and prospects. *Social Network Analysis and Mining*, 1(1):21–26, January 2011.

[26] John P. Scott. *Social Network Analysis: A Handbook.* SAGE Publications, January 2000.

[27] Liaquat; Davis Joseph Chung, Kenneth K S; Hossain. Exploring sociocentric and egocentric approaches for social network analysis. volume NA. NA, online proceedings, 2005.

[28] A. R. Radcliffe-Brown. On Social Structure. *The Journal of the Royal Anthropological Institute of Great Britain and Ireland*, 70(1):1–12, 1940.

[29] Anatol Rapoport. Spread of information through a population with socio-structural bias: I. Assumption of transitivity. *Bulletin of Mathematical Biology*, 15(4):523–533, December 1953.

[30] M.S. Granovetter. The Strength of Weak Ties. *The American Journal of Sociology*, 78(6):1360–1380, 1973.

[31] Maayan Roth, Assaf Ben-David, David Deutscher, Guy Flysher, Ilan Horn, Ari Leichtberg, Naty Leiser, Yossi Matias, and Ron Merom. Suggesting friends using the implicit social graph. In *KDD*, pages 233–242, 2010.

[32] Ruben K. Dagda, Jianhui Zhu, Scott M. Kulich, and Charleen T. Chu. Mitochondrially localized erk2 regulates mitophagy and autophagic cell stress: implications for parkinson's disease. *Autophagy*, 4(6):770–782, Aug 2008.

[33] Yoko Kimura and Keiji Tanaka. Regulatory mechanisms involved in the control of ubiquitin homeostasis. *J Biochem*, 147(6):793–798, Jun 2010.

[34] John McCrae and Nigel Collier. Synonym set extraction from the biomedical literature by lexical pattern discovery. *BMC Bioinformatics*, 9:159, 2008.

[35] Peter Schattner. Cambridge University Press, 2008.