Otto–von–Guericke–Universität Magdeburg

Studienarbeit

Ranking and importance in complex networks

von

Florian Knorn
(* xxx Dez. 1981 in Berlin)

3. Oktober 2005

Eingereicht an die: Otto–von–Guericke–Universität Magdeburg
Fakultät für Elektrotechnik und Informationstechnik
Institut für Automatisierungstechnik
Universitätsplatz 2
Postfach 4120, 39016 Magdeburg
Deutschland

Prüfer: Prof. Jörg Raisch

Betreuer: Dr. Oliver Mason
Prof. Robert Shorten

Hamilton Institute
National University of Ireland
Maynooth, Co. Kildare
Ireland

Meiner Familie

Table of contents

Outline and objectives

In the recent past, there has been enormous interest across a number of research communities in the analysis of complex networks. One of the major motivations for this is the importance of such networks in a variety of diverse application areas, including power systems, the Internet and World Wide Web (WWW), social networks, transportation networks, food webs and biological networks. Furthermore, it was recently established in a sequence of papers that the random graph models, introduced by Erdös and Rényi, traditionally used to model complex networks do not accurately capture many of the key properties of networks that occur in the real world. This led to the introduction of several new network models including the small–world model of Watts and Strogatz (WS) and the scale–free models of Barabási and Albert (BA). A key issue in the analysis of complex networks in several fields is how to identify critical or important nodes within a network. One familiar example of this is provided by the various algorithms developed to rank pages on the WWW in terms of their importance or relevance to a specific search query; the most famous such algorithm being the PageRank algorithm which underpins the Google search engine. A novel interesting application of similar ideas has arisen in the analysis of biological networks such as protein–protein interaction and transcriptional networks, where algorithms for the identification of critical genes or proteins have potential applications in drug design.

This project has a number of objectives. Firstly, the student should become familiar with the basics of Graph Theory and the main statistics used to characterize graph structure as well as the major algorithms used to rank pages on the WWW; namely the HITS algorithm of Kleinberg and the PageRank algorithm of Brin and Page. Specifically, the student should be able to implement these algorithms in MATLAB or an appropriate programming language, and explain clearly the theory behind the algorithms and their convergence properties. The student should also be able to describe a number of other measures of importance in networks including damage, status and excentricity. Secondly, the student should survey the recent models of complex networks, be able to describe and quantify the main properties of these models, and should produce programs to generate and analyze the major network models introduced in recent years. The third strand of the project relates to the application of ranking algorithms to networks subject to uncertainty. This is of particular relevance in biological applications where network data is always considerably noisy. The student should quantitatively assess the sensitivity of PageRank and HITS to various types of uncertainty in network structure, and compare the sensitivity of these measures to that of other traditional notions of network importance such as status.

Preface

Think where man's glory most begins and ends,
And say my gloray was I had such friends.

The Municipal Gallery Re–Visited
William B. Yeats

Acknowledgments

There are many people who contributed to the effort that is my *Studienarbeit*. Without their generous assistance it would not have been possible to write it.

Firstly I would like to express my deep gratitude to Prof. Jörg Raisch. He established the connection to the *Hamilton Institute* and arranged my stay here. In all the years at university I have seen him as my mentor and I would like to thank him for his great teaching, advice and continuing support throughout the years.

Secondly I would like to thank in part the *Studienstiftung des deutschen Volkes* ("German National Academic Foundation") and the *Hamilton Institute* (ultimately the *Science Foundation Ireland*, grant number `04/In1/I478`) for their generous financial support. Without that, it would have been very hard for me to afford the six months I spent here in Maynooth.

Next, I'd like to thank Prof. Robert Shorten from the *Hamilton Institute* for taking a chance on an unknown young man from Germany and being mostly convinced that I could do this. His wise advice and experience he shared extending way beyond aspects related to this Studienarbeit were a great honour to receive and follow, not to forget the cracking basketball games at the end of the weeks.

The person, however, I worked most closely with and who invested a lot of time and thoughts in me is Dr. Oliver Mason. It will be hard to leave him, his sagacious advice as well as his humor behind, and I thank him deeply for all his help, support, ideas and patience.

I also feel deeply honoured and grateful for the opportunity to work at the *Hamilton Institute*. It is such a high–profile yet friendly and open minded environment that I will miss a lot. It was a pleasure to sit in the countless seminars held by many impressive international scientists. Furthermore I am most thankful for the many opportunities for my future that have arisen from my stay in the institute.

Moreover, I would like to thank the following people, colleagues, office mates or Erasmus students for their help and for being many welcome sources of dis-

traction, adding some "live" to the "work" and having those many cool parties. In alphabetical order with titles omitted: Amandine Mukeka, Anat Zil–Bar, Anthony Ng, Barak Pearlmutter, Baruch Even, Berenice Sanchez, Camille Hyron, Carlos Villegas, Caroline Terrisse, Cornelia Nell, David Malone, Delphine Wibaux, Fabian Wirth, Ian Dangerfield, Ian Robertson, Kai Wulff, Kate Moriarty, Laura Karsties, Lea Steppacher, Livia Ruiz, Luke O'Shaughnessy, Mark Verwoerd, Marta Barreras, Mary Quirke, Mehmet Akar, Peter Wellstead, Rade Stanojevich, Richard Middleton, Rosemary Hunt, Santiago Jaramillo, Selim Solmaz, Steven Strachan, Tianji Li, Ulf Schaper, Zhangping Du, ... and many, many more.

I would also like to thank my favorite pub in Maynooth, the *Roost*, for the nice atmosphere for having all those *Pints o' Guinness*, *Jameson* or *Bushmills* Whiskeys, the little dancing area upstairs and the *pub grub* served. And thanks to Amandine's great Salsa classes twice a week we always had a great time those Sunday nights at the *SamSara* in Dublin.

Finally, I could never overstate my gratitude toward my family and close friends for their continuous help and moral support, especially during the darker moments at the beginning of my stay here.

* * *

Declaration of originality

I hereby declare that this thesis and the work reported herein was composed and originated entirely by myself. Information derived from the published and unpublished work of others is acknowledged in the text and a list of references is given in the bibliography.

Magdeburg, 7. Oktober 2005

Florian Knorn

Introduction

Graphs are a very versatile and powerful means of modelling complex, "networklike" systems. This rather abstract mathematical construct can be used to capture interactions between a large number of components or agents, and, beyond allowing for an intuitive graphical representation, provides us with many methods to analyse and manipulate the system we are interested in.

We will use the first chapter to present and review basic notions from graph theory to have a clear framework to build our further investigations on.

Graph theory is an established discipline in discrete mathematics, but until fairly recently only relatively small graphs have been considered. However, advances in many fields have brought along tremendously more complex and larger graphs. For instance, the concept of a graph has been used to describe networks as large and diverse as the internet, neural networks of living creatures, telephone networks or social relations.

As in most other disciplines people tried to create models that reflect or imitate the properties of real world systems. The most prominent and seminal of these is certainly the classical random graph model by Erdös and Rényi.

But this rather artificial model cannot capture many of the remarkable aspects of real world networks. The urging need for better models has been satisfied to some extent by the introduction of more complicated models in the recent past, like the small–world model by Watts and Strogatz, or the scale–free model by Barabási and Albert. The second chapter of this document focuses on these three graph models. We shall shed some light on their characteristics as well as on how they compare with each other, and with real world networks.

The increasing complexity and amount of data becoming available from real world applications but also randomly generated networks brought along pressing needs for methods and algorithms to analyse them. This can involve determining global properties of the network, i. e. characterising the network as a whole, but also local or individual measures for groups of nodes or individual nodes.

A very common task would be to identify important or critical nodes of a network. An important application is the retrieval of relevant information from a knowledge base in the form of a large network, a very prominent example being the world wide web. Finding the right piece of information then boils down to finding important or relevant nodes relative to the query. The knowledge of key nodes could also be used to either target them, as one would want to do for example in networks describing the spread of epidemics — or protect them, like in networks representing some sort of infrastructures for instance.

It is the aim of the third chapter to review some of the highly ingenious ranking schemes that try to answer the need for identifying important nodes

in a network. These include the famous PageRank algorithm, the fairly recent measure called "damage", but also a number of classical topological measures that have been established in relation with resource allocation problems.

Networks drawn from real world systems are usually based on some sort of physical measurement which in cases may introduce considerable amounts of noise. Before using a ranking scheme it is crucial to evaluate its sensitivity to wrong, incomplete, missing or uncertain data.

This is especially important in domains like biology, where one has to face significant amounts of noise in the data due to numerous reasons. A scheme that inverts the order of the nodes in the ranking when only a few edges are changed would clearly be of little practical use.

For that reason, we shall evaluate in the fourth chapter the robustness of the introduced ranking schemes with respect to perturbations in the network. We establish empirical ways of measuring deviations between rankings and use these to analyse the sensitivity of the ranking schemes on randomly generated scale–free graphs.

Another question that needs to be addressed is the actual usefulness of the rankings, i. e. the ability of the ranking schemes to actually identify essential nodes with respect to certain criteria (based on the application). Obviously, this has to be done in conjunction with real data where we *know* which nodes are important and which are not.

We evaluate this on biological data in Chapter 5 using three different protein–protein interaction networks that have been established recently. These datasets provide us with the networks as well as the set of nodes that are know to be essential to the survival of the organism. In particular, we investigate the connection between the essentiality of nodes and the amount of importance attributed to them by the different ranking schemes.

All the algorithms used in the preparation of this document have been written in MATLAB and are available on the accompanying web page of this document, [36]. Most of them are commented on and explained in the text, together with the core of their code. We used the freely available program PAJEK, [7], to generate the graphics. The three datasets from Chapter 5 can also be found in [65].

CHAPTER 1

Notions from Graph theory

A few basic notions from graph theory will be recalled, introducing various definitions and terminology, four types of matrices associated with graphs and some characteristic values.

1.1 Introduction

The notion of a "graph" as currently used in graph theory was first introduced in the first half of the eighteenth century by the swiss mathematician Leonhard Euler, who tried to solve the *Königsberg bridge problem*[1]. It is said that about 100 years later the english mathematician James J. Sylvester coined the word "graph" as we currently know it.

A surprisingly large number of systems have a complex, weblike structure which can be described using graphs. So clearly, results from graph theory allow investigation — and explanation — of many properties of these systems.

Examples can be drawn from many aspects of life, [2]. Just to name a few, large graphs have been used to describe the hyperlink structure of the world wide web, the system of routers and computers forming the internet, complex chemical reaction networks in a cell, steps in protein foldings, neural networks, power grids, cellular and phone networks, collaboration networks of scientists or actors, word occurrences or patterns in linguistics, power grids, transportation or traffic networks.

This chapter discusses several basic notions from graph theory to facilitate our later discussion of random graphs.

1.2 Basic definitions

1.2.1 Graphs

Let's start off with a very general definition of a graph:

Definition 1.1 (Graph)

A ***graph*** $\mathcal{G}$ *is a couple of finite sets* $(\mathcal{V}, \mathcal{E})$*, where* $\mathcal{V}$ *is the set of* ***nodes****, and* $\mathcal{E}$ *is the set of* ***edges*** *between the nodes,* $\mathcal{E} \subseteq \big\{(u,v)\big|u,v \in \mathcal{V}\big\}$.

[1] At the time, the city of Königsberg consisted of two islands in a river, linked by seven bridges. On his morning walks, Euler wondered if there was a route beginning and ending at the same point and traversing all the brigdes exactly once. To find out whether this was possible or not, Euler modeled the problem with a graph ...

Usually, one distinguishes between two major types of graphs: directed graphs (also called digraphs) and undirected graphs.

A graph is said to be undirected if the adjacency relation defined by the edges is symmetric (i. e. $\mathcal{E} \subseteq \big\{\{u,v\}\big|u,v \in \mathcal{V}\big\}$ is made up of *sets* of nodes rather than *ordered pairs*). Otherwise the graph is called directed.

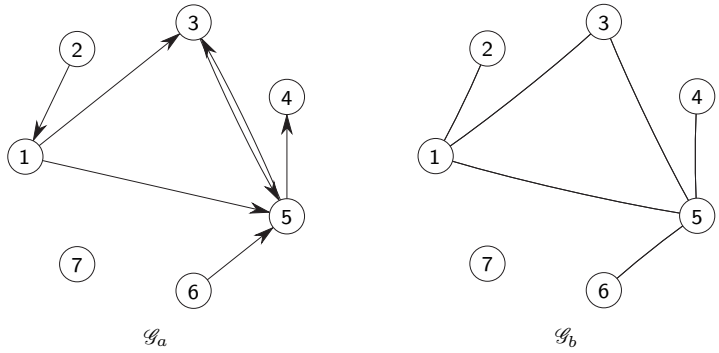
An example of a directed graph $\mathscr{G}_a = (\mathcal{V}_a, \mathcal{E}_a)$ as well as the corresponding undirected graph $\mathscr{G}_b = (\mathcal{V}_b, \mathcal{E}_b)$ is shown in Figure 1.1.[2] Their vertex sets are

$$\mathcal{V}_a = \mathcal{V}_b = \{1,2,3,4,5,6,7\}$$

and their edge sets are

$$\mathcal{E}_a = \big\{(1,3),(1,5),(2,1),(3,5),(5,3),(5,4),(6,5)\big\}$$
$$\mathcal{E}_b = \big\{\{1,3\},\{1,5\},\{2,1\},\{3,5\},\{5,4\},\{6,5\}\big\}$$

Usually, graphs are depicted using dots or circles for the nodes; pairs of them may be joined by a line if the corresponding nodes are connected. Arrow tips at the end of a line usually indicate that the edge is directed.



$\mathscr{G}_a$ $\mathscr{G}_b$

Figure 1.1: *Digraph $\mathscr{G}_a$ and the corresponding undirected graph $\mathscr{G}_b$.*

1.2.2 Edges and nodes

Two nodes (or vertices or points) of a graph may be connected by an edge (or line or branch), corresponding to one element of the edge set. Directed edges are often called arcs.

An edge has either one or two nodes associated with it, called endpoints. An edge is said to join its endpoints. If these endpoints are distinct, the edge joining them is called proper; an edge joining a single endpoint to itself is called a self–loop. A multi–edge is a collection of two or more edges having identical endpoints.

If a node v is an endpoint of an edge e, then e is said to be incident on v and v is incident on e. If two nodes u and v are joined by an edge, u and v are said to be adjacent or connected and usually called neighbours. A node u is said to be reachable from v if there exists an (un)directed path with v as initial node and u as terminal node.

[2] Graphics created with scPajek (http://vlado.fmf.uni-lj.si/pub/networks/pajek/).

A node can be connected to no other nodes, one node, or many nodes, including itself. The number of edges attached attached to a node is called its degree (or valence). In directed graphs, one may further distinguish between the indegree and the outdegree of a node (as being the number of in– respectively outgoing edges to respectively from it). A node with degree 0 is called isolated.

1.2.3 Special graphs

To be able to define some further properties of graphs, we need the definitions of "paths" and "cycles":

Definition 1.2 (Paths and cycles)

Let $\mathscr{G} = (\mathcal{V}, \mathcal{E})$ be a graph, $m \in \mathbb{N}$ and $v_i \in \mathcal{V}$, $i = 1, \ldots, m+1$, are $m+1$ distinct vertices. If exists a sequence $\varphi \in \mathcal{E}^m$ with

$$\varphi = \begin{cases} \big((v_1, v_2), (v_2, v_3), \ldots, (v_m, v_{m+1})\big) & \text{if } \mathscr{G} \text{ is directed} \\ \big(\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_m, v_{m+1}\}\big) & \text{if } \mathscr{G} \text{ is undirected} \end{cases}$$

*it is called a (directed) **path** of length m. If $v_1 = v_{m+1}$, the path is called **closed**. The shortest path between two distinct nodes i and j is called **geodesic**. Its length is denoted l_{ij} and often referred to as **distance** between i and j.*

*A **cycle** (or **circuit**) is a closed path with length ≥ 3 and no repeated internal nodes.*

With these basic notions we can now review a few special, but classical types of graphs. A graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$ is said to be

- simple if it contains no multi–edges and no self–loops,
- bipartite if there exists a non–trivial partition $\mathcal{V} = \mathcal{V}^+ \cup \mathcal{V}^-$ of the vertex set with $\mathcal{V}^+ \cap \mathcal{V}^- = \emptyset$ such that for all edges $(u, v) \in \mathcal{E}$ (or $\{u, v\} \in \mathcal{E}$) $u \in \mathcal{V}^+$ and $v \in \mathcal{V}^-$,
- complete if there exists one and only one edge between every pair of distinct nodes,
- an underlying graph if it is the undirected version of a directed graph,
- connected if every node can be reached from every other node (in the undirected case). If this holds for a digraph, it is called strongly connected, if it only holds for its underlying undirected graph, it is said to be weakly connected,
- (k–)regular if all its nodes have the same degree (k),
- a subgraph of another graph $\tilde{\mathscr{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ if $\mathcal{V} \subseteq \tilde{\mathcal{V}}$ and $\mathcal{E} \subseteq \tilde{\mathcal{E}}$.

Note: In the following we will only focus on *simple* graphs as most of the graphs we will be encountering in this paper are simple graphs!

Furthermore we will always suppose that a graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$ we are investigating has $n = |\mathcal{V}|$ nodes and that they are numbered and explicitly ordered $1, \ldots, n$.

1.3 Matrices associated with graphs

In order to work with graphs, the representation using two sets is not very useful. In fact, all the information contained within these two set (with the exception of the names of the nodes, if they are not explicitly numbered 1 to n) can be contained in a matrix.

1.3.1 Adjacency matrix

One of the most common — and probably the most intuitive matrix associated with graphs is the following.

Definition 1.3 (Adjacency matrix)
The $n \times n$ ***adjacency matrix*** $\boldsymbol{A}$ *of a graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$ is defined as*

$$a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in \mathcal{E} \text{ resp. } \{v_i, v_j\} \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

and hence is symmetric if $\mathscr{G}$ is an undirected graph.

The adjacency matrices for the two graphs from Figure 1.1 on page 2 would be the following:

$$\boldsymbol{A}_{\mathscr{G}_a} = \begin{bmatrix} \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \quad , \quad \boldsymbol{A}_{\mathscr{G}_b} = \begin{bmatrix} \cdot & 1 & 1 & \cdot & 1 & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ 1 & \cdot & 1 & 1 & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

where " $\cdot$ " corresponds to a zero entry (for better legibility). As in this example we will sometimes index adjacency matrices with the names of the graphs they correspond to (to avoid any ambiguity).

Note that the last row and column in both matrices is all empty, corresponding to the isolated node 7. Furthermore, as we have no self–loops, all main diagonal elements are also all zero.

Connected graphs

In the example above, as node 7 cannot be reached from any other node, $\mathscr{G}_b$ is not connected. The property of a graph being either connected or not is closely related to a certain property of its adjacency matrix.

It's a well know fact, and stated formally in Theorem A.1 on page 97 in the Appendix, that a graph is (strongly) connected if and only if its adjacency matrix is irreducible in the sense of the following definition.

Definition 1.4 (Irreducibility of a matrix)

*A matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ with is said to be **reducible** if either*

(i) $n = 1$ and $\boldsymbol{A} = 0$; or

(ii) $n \geq 2$, there is a permutation matrix $\boldsymbol{\Pi}_n \in \mathbb{R}^{n\times n}$ and there is some integer r with $1 \leq r \leq n-1$ such that

$$\boldsymbol{\Pi}_n^T \boldsymbol{A} \boldsymbol{\Pi}_n = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{0} & \boldsymbol{D} \end{bmatrix} \tag{1.2}$$

where $\boldsymbol{B} \in \mathbb{R}^{r\times r}$, $\boldsymbol{D} \in \mathbb{R}^{(n-r)\times(n-r)}$, $\boldsymbol{C} \in \mathbb{R}^{r\times(n-r)}$ and $\boldsymbol{0} \in \mathbb{R}^{(n-r)\times r}$ is the zero matrix.

Note that this definition does not require the blocks $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ to have nonzero entries, but only that one should be able to get an $(n-r)$–by–r block of 0 entries in the indicated position by some sequence of row and column interchanges.

Powers of the adjacency matrix

Looking at the adjacency matrix $\boldsymbol{A}$ of a graph with n nodes, we basically see the locations of paths of length 1 between each pair of nodes ($a_{ij} = 1$ per definition means there is an edge between node i and j). Strictly speaking, we see the *number* of paths of length 1 between each pair of nodes, as we will see now.

It is a fact that if we look at a positive power of the adjacency matrix, $\boldsymbol{A}^m$, $m \in \mathbb{N}$, each entry a_{ij}^m tells us the number of paths of length m between node i and node j.[3]. This can easily be seen if we recall that

$$a_{ij}^2 = \sum_{k=1}^{n} a_{ik} a_{kj}$$

Here, the value of a_{ij}^2 corresponds to the number of cases where there are two edges (i,k) and (k,j), i. e. the number of different paths of length two that go from node i to node j. This can then be easily extended to higher powers of $\boldsymbol{A}$.

With this in mind, it is quite immediate to see that a matrix $\boldsymbol{A}$ is irreducible if and only if some positive power of $(A + \mathbf{id})$ is strictly positive (the addition of the identity matrix is needed to ensure the fact that every node is reachable from itself).

[3] a_{ij}^m denotes element (i,j) of $\boldsymbol{A}^m$, see also the List of Symbols on page 110.

1.3.2 Laplacian matrix

Another commonly used matrix for analyzing graphs is the Laplacian matrix, which we now define.

Definition 1.5 (Laplacian matrix)

*For an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes, its $n \times n$ **Laplacian matrix** (also called **graph Laplacian**, just **Laplacian**, **Admittance**– or **Kirchhoff matrix**) $\boldsymbol{L}_\mathcal{G}$ is defined as the difference between its degree matrix and its adjacency matrix:*

$$\boldsymbol{L}_\mathcal{G} = \boldsymbol{D}_\mathcal{G} - \boldsymbol{A}_\mathcal{G}$$

where $\boldsymbol{D}_\mathcal{G} = \mathrm{diag}(k_1, k_2, \ldots, k_n)$, k_i being the degree of node i.

To illustrate this definition, the Laplacian of $\mathcal{G}_b$ from Figure 1.1 on page 2 would be

$$\boldsymbol{L}_{\mathcal{G}_b} = \begin{bmatrix} 3 & -1 & -1 & \cdot & -1 & \cdot & \cdot \\ -1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ -1 & \cdot & 3 & \cdot & -1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & -1 & \cdot & \cdot \\ -1 & \cdot & -1 & -1 & 4 & -1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & -1 & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

The definition of the Laplacian allows for it to have a couple of important and elementary properties:

Proposition 1.1 (Positive semidefiniteness of the Laplacian)

For an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ the following hold for its Laplacian:

(i) $\boldsymbol{L}_\mathcal{G}$ *is positive semidefinite.*

(ii) The smallest eigenvalue of $\boldsymbol{L}_\mathcal{G}$ is $\lambda_1 = 0$ and a corresponding eigenvector is $\mathbf{1}_n$, the n–element column vector with all ones.

(iii) Let $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ be the eigenvalues of $\boldsymbol{L}_\mathcal{G}$. If $\mathcal{G}$ is connected, then $\lambda_2 > 0$.

Proof: Partly taken from [58]:

(i): Let $\mathcal{G}_{1,2}$ denote the graph with two nodes and one (undirected) edge connecting them (•—•). Then, its Laplacian is

$$\boldsymbol{L}_{\mathcal{G}_{1,2}} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

Obviously, $\boldsymbol{x}^T \boldsymbol{L}_{\mathcal{G}_{1,2}} \boldsymbol{x} = (x_1 - x_2)^2 \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^2$. As every (non–empty, simple and undirected) graph $\mathcal{G}$ is made up a multitude of these $\mathcal{G}_{1,2}$'s, it is clear that

$$\boldsymbol{x}^T \boldsymbol{L}_\mathcal{G} \boldsymbol{x} = \sum_{(u,v) \in \mathcal{E}} (x_u - x_v)^2 \geq 0$$

for all $\boldsymbol{x} \in \mathbb{R}^n$.

(ii): As the sum over a row from $\boldsymbol{A}_{\mathscr{G}}$ is equal to the degree of the corresponding node, the sum over a row of $\boldsymbol{L}_{\mathscr{G}}$ is always zero. Hence $\boldsymbol{L}_{\mathscr{G}}\mathbf{1}_n = \mathbf{0}_n$.

(iii): Let $\boldsymbol{\nu}$ be an eigenvector of $\boldsymbol{L}_{\mathscr{G}}$ of eigenvalue 0. Then $\boldsymbol{L}_{\mathscr{G}}\boldsymbol{\nu} = \mathbf{0}_n$ and

$$\boldsymbol{\nu}^T \boldsymbol{L}_{\mathscr{G}} \boldsymbol{\nu} = \sum_{(u,v)\in\mathcal{E}} (\nu_u - \nu_v)^2 = 0$$

Thus for each pair of nodes (u, v) connected by an edge, we must have $\nu_u = \nu_v$. As the graph is connected, we have $\nu_u = \nu_v$ for *all* pairs of nodes (u,v), which implies that $\boldsymbol{\nu}$ is a scalar multiple of $\mathbf{1}_n$. Thus, the eigenspace associated with $\lambda_1 = 0$ has dimension 1, and since $\boldsymbol{L}_{\mathscr{G}}$ is positive semidefinite, λ_2 must be strictly positive. □

As a result from linear algebra, the fact that $L_{\mathscr{G}}$ is always positive semidefinite implies that all its eigenvalues are real and nonnegative, which was used in the proof of (iii).

This property has actually led Fiedler to consider the magnitude of λ_2 as a measure of the connectedness of a graph [21], calling λ_2 the algebraic connectivity. Loosely speaking: the greater λ_2, the more "connected" the graph.

As the spectrum of a graph is made of the union of the spectra of its connected components, the multiplicity of 0 as an eigenvalue of $\boldsymbol{L}_{\mathscr{G}}$ is equal to the number of connected components[4].

A number of different results using λ_2 can found in [47], like bounds for the diameter, average path length or other characteristic values of a graph, [46]. These are useful for extremely large networks that are out of scope of algorithms like those shown in the next section.

Before moving on, let's have a quick look at our example from Figure 1.1 on page 2 again. There, the three smallest eigenvalues of $\boldsymbol{L}_{\mathscr{G}_b}$ are 0, 0, and $\simeq$ 0.6314, indicating, that it has two connected components (that is the connected nodes 1 to 6 on the one hand, as well as the isolated node 7 on the other).

1.3.3 Distance matrix

Another matrix we are going to look at is the distance matrix. It contains all the distance information of the graph, that is the shortest path lengths between any two (distinct) nodes.

We present here a somewhat customized definition of the distance matrix, allowing us to extend its classical definition to disconnected graphs.

Definition 1.6 (Distance matrix)

For a given graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$, its $n \times n$ ***distance matrix*** $\boldsymbol{\Delta}$*. is given by*

$$\delta_{ij} = \begin{cases} 0 & \text{if } i = j \\ l_{ij} & \text{if node } i \text{ can be reached from node } j \\ n & \text{if node } i \text{ cannot be reached from node } j \end{cases}$$

[4] A connected component of a graph $\mathscr{G}_1$ is a connected subgraph $\mathscr{G}_2$ such that no subgraph of $\mathscr{G}_1$ that properly contains $\mathscr{G}_2$ is connected. In other words, a connected component is a maximal connected subgraph.

where $n = |\mathcal{V}|$ is the total number of nodes in the graph.

The usual definition of the distance matrix either demands connectedness, or, if not, sets the corresponding entries for disconnected nodes to infinity. For our applications later we choose not to set it to infinity, as this does not allow for any quantification to which extent a node is disconnected. This will be particularly important when we look at centrality measures like excentricity, status and centroid value (see Section 3.5 on page 52).

Obviously, this matrix will be symmetric in the case of undirected graphs. One way of calculating it is mentioned in Footnote 7 on page 11.

In our example from Figure 1.1 on page 2, the distance matrix of the directed graph $\mathscr{G}_a$ is:

$$\boldsymbol{\Delta}_{\mathscr{G}_a} = \begin{bmatrix} \cdot & 7 & 1 & 2 & 1 & 7 & 7 \\ 1 & \cdot & 2 & 3 & 2 & 7 & 7 \\ 7 & 7 & \cdot & 2 & 1 & 7 & 7 \\ 7 & 7 & 7 & \cdot & 7 & 7 & 7 \\ 7 & 7 & 1 & 1 & \cdot & 7 & 7 \\ 7 & 7 & 2 & 2 & 1 & \cdot & 7 \\ 7 & 7 & 7 & 7 & 7 & 7 & \cdot \end{bmatrix}$$

1.3.4 Incidence matrix

For reasons of completeness, let's define a third type of matrix usually associated with an undirected graph [47]:

Definition 1.7 (Incidence matrix)

*For a given directed graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$, the $|\mathcal{V}| \times |\mathcal{E}|$ **incidence matrix** $\boldsymbol{I}$ is defined by*

$$i_{ve} = \begin{cases} -1 & \text{if edge } e \text{ is directed } \textbf{to} \text{ node } v \\ 1 & \text{if edge } e \text{ is directed } \textbf{from} \text{ node } v \\ 0 & \text{otherwise} \end{cases}$$

In case of an undirected graph, first orient its edges arbitrarily, i. e. for each $e \in \mathcal{E}$, choose one of its endpoints as the initial node, and the other as terminal node.

Defining the incidence matrix for an undirected graph as above we have

$$\boldsymbol{L}_{\mathscr{G}} = \boldsymbol{I}_{\mathscr{G}} \boldsymbol{I}_{\mathscr{G}}^T \tag{1.3}$$

independent of the orientation chosen for the edges [9]. This property can be used for an alternative proof of Propos. 1.1(i) on page 6. Using the incidence matrix and the scalar product, we can write with Equation (1.3)

$$\boldsymbol{x}^T \boldsymbol{L}_{\mathscr{G}} \boldsymbol{x} = \langle \boldsymbol{L}_{\mathscr{G}} \boldsymbol{x}, \boldsymbol{x} \rangle = \langle \boldsymbol{I}_{\mathscr{G}} \boldsymbol{I}_{\mathscr{G}}^T \boldsymbol{x}, \boldsymbol{x} \rangle = \langle \boldsymbol{I}_{\mathscr{G}}^T \boldsymbol{x}, \boldsymbol{I}_{\mathscr{G}}^T \boldsymbol{x} \rangle \geq 0$$

showing that the Laplacian of a graph is always positive semidefinite.

For our small graph $\mathscr{G}_a$ from Figure 1.1 on page 2, the incidence matrix would be:

$$\boldsymbol{I}_{\mathscr{G}_a} = \begin{bmatrix} -1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & -1 & \cdot & 1 & -1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot \\ \cdot & \cdot & -1 & -1 & 1 & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

Now that we gathered the necessary terminology we should take a look at some characteristic values associated with graphs.

1.4 Characteristic values

To be able to better classify and compare the topologies of the random networks encountered in the next chapter we need a number of local and global measures associated with networks.

As we could not find any practical implementations of algorithms that calculate or determine those measures — nor good hints on how to realise them — we had to come up with an exact way of calculating them as well as sufficiently fast implementations for our purposes. For both reasons, we present with each value the core routines of the MATLAB programs that accompany this paper.[5]

1.4.1 Average node degree

As the name suggests, the definition is straightforward:

Definition 1.8 (Average node degree)

*The **average node degree** of an undirected graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$ is defined as*

$$\langle k \rangle = \frac{1}{n} \sum_{i=1}^{n} k_i$$

where k_i is the degree of node i and $n = |\mathcal{V}| \geq 1$ is the total number of nodes.

If $\mathscr{G}$ is a directed graph, then

$$\langle k \rangle = \frac{1}{n} \sum_{i=1}^{n} \left(k_{\mathrm{in}i} + k_{\mathrm{out}i} \right)$$

where $k_{\mathrm{in}i}$ and $k_{\mathrm{out}i}$ are the in– and outdegrees of node i.

The MATLAB code for this value is also straightforward. One can easily see that $\langle k \rangle = 2e/n$, where $e = |\mathcal{E}|$ is the total number of edges. So in the m–file,

[5] Implementations in lower level programming languages may differ significantly from the approaches taken here, where we tried to avoid loops as much as possible and rather used matrix operations and built in routines of MATLAB.

we only need to get the number of non–zero entries in the adjacency matrix, which gives us the number of edges e, and divide by n, if we're dealing with an undirected graph.

In the directed case, however, each edge is not accounted for with *two* entries in the adjacency matrix, so we must not forget to multiply by two. The resulting simple program is shown in Listing 1.1.

```
1 function k = avk( A )
2
3 if A ≠ A' % A is not symmetric => directed graph
4    k = 2*length(find(A))/size(A,1);
5 else      % A is symmetric => undirected graph
6    k = length(find(A))/size(A,1);
7 end
```

Listing 1.1: *Function for determining the average path length of a graph.*

1.4.2 Average path length

Definition 1.9 (Average path length)

The ***average path length*** $\langle l \rangle$ *of a (strongly) connected graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *is defined as the average length of geodesics between every pair of distinct nodes:*

If l_{ij} *is the length of the geodesic between the distinct nodes* i *and* j*, and* $\mathcal{G} = \left\{ l_{ij} \middle| i, j \in \mathcal{V}, i \neq j \right\}$ *is the set of all geodesics, then*

$$\langle l \rangle = \frac{1}{|\mathcal{G}|} \sum_{l \in \mathcal{G}} l$$

If the graph is not connected, the average path length is defined as the average lengths of the geodesics of every pair of mutually reachable nodes, i. e. $\mathcal{G} = \left\{ l_{ij} \middle| i, j \in \mathcal{V}, i \neq j, \text{node } i \text{ can be reached from node } j \right\}$ *in this case.* _

There are basically two different ways of determining $\langle l \rangle$. We could first gather all the lengths of all the needed geodesics using specialized algorithms that find shortest paths (for example Dijkstra's algorithm [17] or the Bellman–Ford algorithm [8, 34]) and then take the average.

However, as it proved to be significantly faster in our setup[6] and provides some further information, we use a rather "brute force" algorithm:

Calculation

Suppose we have a simple and connected graph. We know that powers of the adjacency matrix tell us the number of paths with a certain length between

[6] That is with MATLAB. Here, our algorithm mainly using the built–in matrix operations always outperformed a (MATLAB–) implementation of Dijkstra's algorithm as the latter involves several levels of nested `for`–loops. However, in lower level implementations the opposite will be the case as not only will there be, in average, less operations and memory required, but also loops are evaluated much faster.

each pair of nodes. Hence in order to determine the shortest path between two nodes i and j we can look at powers of the adjacency matrix and wait for the emergence of a non–zero value in the ij–th entry. So if for a certain $\tilde{m} \in \mathbb{N}$ we have $a_{ij}^{\tilde{m}} \neq 0$ where for lower powers that entry was always zero, we can conclude that the shortest path from node i to node j has length $\tilde{m}$.

Effectively, all that needs to be done is keep track of the number of *new* non–zero entries at each step (new power of $\boldsymbol{A}$) and then calculate the average. To do so, we iteratively sum up the powers of $\boldsymbol{A}$:

$$\boldsymbol{A}_{(m)}^{\Sigma} = \boldsymbol{A}^0 + \boldsymbol{A}^1 + \boldsymbol{A}^2 + \ldots + \boldsymbol{A}^m \tag{1.4}$$

As the graph consists of n nodes, the shortest path between two nodes can have at most length $n-1$ (if it was longer, then at least one node has to be "visited" more than once which cannot result in the *shortest* path). For that reason, we only need to test up to $m \leq n-1$ which prevents an infinite loop. If $\boldsymbol{A}_{(\tilde{m})}^{\Sigma}$ is strictly positive for some $\tilde{m} \leq n-1$, then the graph is connected, the longest shortest path has length $\tilde{m}$ and we can stop the iterations.

Another break condition would be if there is no increase in the count of non–zero elements between two steps. This is also an indication that the graph is not connected. In this case, we can stop the iteration and calculate the average even so, *but* we should indicate that the graph is not connected.[7]

In fact, in this case, it is a useful piece of information to know how many connected components the graph is composed of. This can be determined quite easily using $\boldsymbol{A}_{(m)}^{\Sigma}$, as we will see next.

Number of connected components

In order to determine the number of connected components, in case of disconnected undirected graphs, we can use the following iterative process: Taking $\boldsymbol{A}_{(m)}^{\Sigma}$ from Equation (1.4) above that we calculated with the average length algorithm, we first look at its first row. All the zero entries in this row correspond to nodes that cannot be reached from the first node. On the other hand, the number of non–zero entries in that row is the size (as of number of nodes) of the connected component the first node is in, because all these nodes *can* be reached from it.

Now we discard the first connected component by only keeping the rows and columns of $\boldsymbol{A}_{(m)}^{\Sigma}$ where there had been some zero entries. We restart with this new, smaller matrix again by looking at the first row and so on — until we end up with an empty matrix.

The number of times this process can be repeated then corresponds to the number of connected components. This method can also be used find the size of the largest connected component (simply by storing the sizes and finding the

[7] Precisely this procedure can also be used to establish the distance matrix of a graph: at each step, when a new non–zero entry "emerges", simply note its location ij and the value of m (the length of the just "discovered" geodesic between nodes i and j). This way we gather all the information we need to construct $\boldsymbol{\Delta}$.
Using the framework of the algorithm for the average path length at the end of this subsection, all one had to do is to instantiate an empty $\boldsymbol{\Delta}$ with `D = zeros(n,n)` and add the short line `D = D + m*xor(Asum1,Asum2)` at the end of the first `while`–loop, e. g. after Line 16. If the graph is disconnected, we must additionally set all the remaining zero entries (but the ones on the main diagonal) to n, using for example `D(find(¬D))=n; D = D−n*eye(n);`.

maximum) and, of course, determine the connected components themselves (see Subsection 3.6.2 on page 56).

Implementation

We now finish this subsection by presenting the implementation of this technique. Before we move on, a little comment on the handling of sparse matrices in MATLAB is appropriate.

As most of the matrices used in the MATLAB programs here are quite, if not very sparse, dealing with them is much more efficient and faster when we use the `sparse` matrix format rather then using full matrices. That way, MATLAB only calculates where it needs to calculate. Most of the MATLAB functions have a built in optimized version of the algorithms they are using, specifically designed for the `sparse` data type (and they use it automatically).[8]

Another advantage of using sparse matrices are the few additional functions available for this data type, like `nnz` for example, which returns the number of non–zero entries.

To further speed up the algorithms, the entries of the matrices are converted to the `logical` data type. That way MATLAB only has to perform binary operations like `OR`, `NOT` or `AND` on the entries.

With all this in mind, it is now easy to understand our way of implementing the algorithm in Listing 1.2 on the facing page. Taking an adjacency matrix $\boldsymbol{A}$, the function returns of course the average path length $\langle l \rangle$ as well as the number of connected components n_{cc} and the size of the largest connected component n_{lcc}.

1.4.3 Diameter

The following definition reflects the most widely spread interpretation of the diameter of a graph.

Definition 1.10 (Diameter)

The ***diameter*** *d of a (strongly) connected graph is defined as the length of the longest geodesic, i. e. the length of the longest* shortest path.

If the graph is not connected, it is defined the same way, but only for all pairs of mutually reachable nodes.

It is important to mention that some people understand the diameter as the average path length. As this might lead to confusion, we stick to the most popular interpretation and keep the average path length separate.

We can use exactly the same algorithm as for the average path length to calculate the diameter. There, we find the diameter to be the value of m or $m - 1$, depending on why the `while`–loop in Line 11 quit:

– If the loop stopped because $\boldsymbol{A}$ became strictly positive, then `d = m`

[8] For instance, if MATLAB was to calculate the scalar product of two all zeros vectors in the `full` data type, it would first blindly multiply all the corresponding entries and sum them up only to find that the result is zero. If the `sparse` format had been used, it would come up with zero right away, not doing a single multiplication, because it "knows" there is no point in multiplying something by zero (as zero is the result right away) or adding zero to something.

```
function [ l , ncc , nlcc ] = avl( A )

n = size(A,1); m = 0; l = 0; count = []; Δnnz = 42;

% start the testing-loop
A = logical(A);            % make sure, A is of type 'logical'
Asum2 = logical(speye(n)); % sparse identity matrix
Am = Asum2;                % corresponds to A^0
warning('off','MATLAB:conversionToLogical'); % we know that issue.

while (nnz(Asum2) ≠ nn) && (Δnnz ≠ 0) && (m ≤ n)
   m = m + 1;              % increase counter
   Am = logical(double(Am)*A); % raise A to a new power
   Asum1 = Asum2;          % store previous Asum
   Asum2 = Asum1 | Am;     % add new power
   Δnnz = nnz(Asum2)-nnz(Asum1); % increase of new shortest paths
end

% finish by calculating the average
for i=1:length(count)
   l = l + count(i)*i;
end
l = l/sum(count);

% if graph is not connected, count ncc and determine the slcc
if (Δnnz == 0) || (d == n)
   ncc = 0;                % number of connected components
   scc = [];               % size of each connected component
   while ¬isempty(Asum2)
      ncc = ncc + 1;       % increase count of cc
      zeroz = find(Asum2(1,:)==0);% grab zero entries in first row
      scc(ncc) = length(find(Asum2(1,:)>0)); % scc=nnz(first row)
      Asum2 = Asum2(zeroz,zeroz); % Asum2 = all-zero-rows and cols
   end

   nlcc = max( scc );      % find largest connected component
   if isempty(nlcc)        % if it's empty (i.e. only one node)
      nlcc = 1;            % set size to 1
   end

else % graph is connected
   ncc = 1;
   nlcc = n;
end
```

Listing 1.2: *Function calculating average path length, the number of connected components as well as the size of the largest connected component.*

- Else (if Δ`nnz == 0` or `m == n` or), then `d = m - 1` (because the loop went one iteration too far)

1.4.4 Clustering coefficient

Many types of networks have some inherent tendency to form clusters. Within a circle of friends, for example, it is somewhat likely that two friends of somebody are also friends with each other.

One way of measuring the extent of this "cliquishness" was proposed by Watts and Strogatz in [62], where they introduced the clustering coefficient as a measure on how close the neighbourhood of each node comes on average to being a complete subgraph:

Definition 1.11 (Clustering coefficient)

*For a node i with neighbourhood $\mathcal{N}_i = \{ v_j \,|\, e_{ij} \in \mathcal{E}\}$, if $|\mathcal{N}_i| = k_i \geq 2$, the **clustering coefficient** c_i is defined as the proportion of links between the k_i nodes within its neighbourhood divided by the number of links that could possibly exist between them:*

$$c_i = \begin{cases} \dfrac{\big|\{e_{jk}\}\big|}{k_i(k_i-1)} & \text{if } \mathscr{G} \text{ is a directed graph} \\[2ex] \dfrac{2\big|\{e_{jk}\}\big|}{k_i(k_i-1)} & \text{if } \mathscr{G} \text{ is an undirected graph} \end{cases} \tag{1.5}$$

with $v_j, v_k \in \mathcal{N}_i$ and $e_{jk} \in \mathcal{E}$ for $i = 1, \ldots, n$.

*The **(average) clustering coefficient** $\langle c \rangle$ of a graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$ is the average of the clustering coefficients of the nodes that have two or more neighbours.*[9]

Calculation

Many people have also interpreted the definition above as the ratio of existing triangles to possible triangles within the neighbourhood of node i and containing node i. Using the adjacency matrix of the graph a triangle containing node i always has the form $a_{ij}a_{ik}a_{jk} \neq 0$.

So in case of an undirected graph, $\boldsymbol{A} = \boldsymbol{A}^T$, one can write immediately:

$$t_i = \sum_{j>k} a_{ij}a_{ik}a_{jk} \qquad \text{and} \qquad c_i = \frac{2t_i}{k_i(k_i-1)}$$

Note that looking at the main diagonal of $\boldsymbol{A}^3$, we find $t_i = a_{ii}/2$: recall, that entries (i, j) of $\boldsymbol{A}^m$ correspond to the number of paths of length m between node i and node j. So the main diagonal of $\boldsymbol{A}^3$ corresponds to the number of paths of length 3 starting from and ending in each node — in other words the number of triangles. This is a fast way of calculating the average clustering coefficient, but one may run out of memory in the calculation of $\boldsymbol{A}^3$. For that reason, we now present a less memory consuming implementation of the algorithm.

Implementation

The function takes the adjacency matrix $\boldsymbol{A}$ as argument and returns the clustering coefficient $\langle c \rangle$, the number of isolated nodes as well as the number of nodes with degree 1, see Listing 1.3 on the next page. Note that the algorithm is only intended for *undirected* graphs.

[9] The word "average" is usually omitted.

We basically do exactly what has been described above, that is systematically fetch the neighbors of nodes and count the edges among them.

As most of the matrices are rather sparsely filled, bluntly iterating over *all* the $j > k$ would involve a lot more operations than needed. To spare from that, we only look within the set of neighbours $\mathcal{N}_i$ of node i for possible $a_{ij}a_{ik}a_{jk} \neq 0$, cf. Line 19.

Cases of nodes with only zero or one neighbours are taken care of in Lines 11–15: in both cases, the respective counter is incremented (which can later be used to report those cases) and the corresponding entry in the `c` vector will be a `NaN`[10].

With this information we can easily calculate the clustering coefficient for each node (where it makes sense) and take the average, Line 28.

```
1  function [ c , k0count , k1count ] = cc( A )
2
3  n = size(A,1); c = []; k0count = 0; k1count = 0;
4
5  % for each of the n nodes:
6  for i = 1:n
7     % step 1: find immediate neighbours nb of node i:
8     nb = find(A(i,:));       % n_i
9     nnb = length(nb);        % |n_i| = k_i = no. of neighbours
10
11    if nnb == 0  % if k_i = 0 or 1 -> cc concept not applicable
12       k0count = k0count + 1; c(i) = NaN; continue
13    elseif nnb == 1
14       k1count = k1count + 1; c(i) = NaN; continue
15    end
16
17    % step 2: count egdes amongst the neighbours of node i
18    edgecount = 0;
19    for j = 1:nnb
20       for k = j+1:nnb       % symmetry saves work: k > j
21          if A(nb(j),nb(k))
22             edgecount = edgecount + 1;
23          end
24       end
25    end
26
27    % step 3: c_i = exist_edges / possib_edges b/w neighbours
28    c(i) = 2*edgecount / (nnb*(nnb-1));
29
30 end
31
32 % finish by calculating average over all non-NaNs
33 c = full(mean( c(find(~isnan(c))) ));
```

Listing 1.3: *Function calculating the (average) clustering coefficient as well as the number of nodes with 0 and 1 neighbours.*

With the implementation of this algorithm we would like to close this chapter of basic notions from graph theory and move on to random graphs.

[10] `NaN` stands for "Not a Number"

CHAPTER 2

Random graphs

We introduce three major types of random graphs, analyse their characteristics and generic properties, and compare their behaviour with some real world examples.

2.1 Introduction

As mentioned earlier, the concept of graphs has been around for some centuries. But until the 1950s, attention was paid only to "regular" graphs. In fact, it was the Hungarians Paul Erdös and Alfréd Réyni who extended the focus on large–scale networks with *no* apparent design principles — "random graphs". Since their famous paper [19] from 1959, random graph theory has become one of the main areas of interest in modern discrete mathematics, producing many and some highly ingenious results. Some of them for instance allow us to better understand the mechanisms that determine or lead to the specific topology of a large network.

Some dramatic advances have been made in the past few years, mainly made possible by increases in computational power as well as the computerisation of data acquisition in many fields allowing for large databases and abundant data of various real networks. On the other hand, as boundaries break down between different disciplines, collaboration of mathematicians, computer experts and biologists, for example, have brought some interesting advances in systems biology, as the classical reductionist modeling approach cannot always give an explanation for the behaviour of a system as a whole.

In the absence of real data, or for simulation purposes, people tried to recreate the phenomena observed in real networks using random networks. As a result, a large variety of graph models has been established.

In this chapter we shall review a number of interesting and important findings not only for the classical Erdös–Rényi model but also for two other, more recent models, namely the one by Watts and Strogatz, and the one by Barabási and Albert. Especially the last model of these will be used extensively in our later work.

For a first, uncommented impression, Figure 2.1 on the following page shows three small graphs with 20 nodes and 20 edges created using those models. Larger examples are can be found in Figure 2.2, Figure 2.5 and Figure 2.8 on pages 21, 27 and 33 respectively, each of them having $n = 100$ nodes and about $e = 200$ edges.

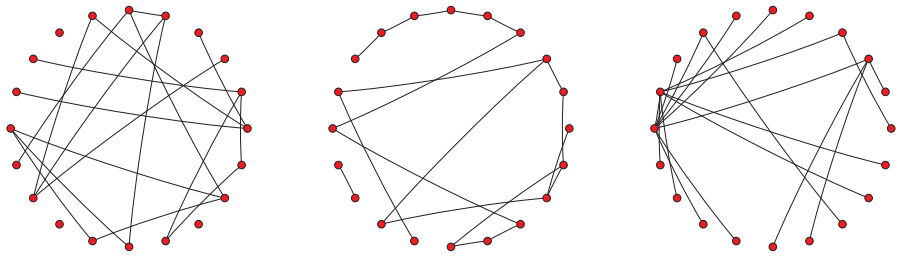


Figure 2.1: *Small instances of our three random graph models:* ***(a)*** *Erdös–Rényi,* ***(b)*** *Watts–Strogatz and* ***(c)*** *Barabási–Albert.*

2.2 The Erdös–Rényi Model

As mentioned above, this is historically the first model of a random graph. It is probably the most natural way of creating a random graph: choose (fix) a number of nodes n and then independently connect each pair of distinct nodes with an equal probability p.

Erdös and Rényi discovered in the late 1940s that certain probabilistic methods were often useful in tackling problems in graph theory. Due to its relatively long history this model has been extensively studied (the Watts–Strogatz and Barabási–Albert models have only been around for less than seven years). A useful resource is Béla Bollobás' excellent book on random graphs, [11].

A typical Erdös–Rényi graph is shown in Figure 2.2(a) on page 21.

2.2.1 Generation of Erdös–Rényi graphs

Generating matrices with random entries in MATLAB is usually very fast. The basic idea of our algorithm, shown in Listing 2.1 on the facing page, is to create a matrix with all random (uniformly distributed) entries, take — as we want an undirected graph — the upper triangular part[1], replace all the entries that are larger than $(1 - p)$ by `true`–entries, the rest by `false`–entries, and finally "mirror" down the upper triangle to create the full, symmetric adjacency matrix.

The only problems may arise in lines 8, 13 and 16, where the generated random matrix `R` can become quite large, thus memory consuming, as it is a full matrix with n^2 `double`–entries (that require 2^3 bits each). For a desired network size of for example $n = 8192 = 2^{13}$ nodes a computer would need at least a total of $2^{13} \cdot 2^{13} \cdot 2^3 = 2^{29}$ bytes or $2^9 = 512$ MB of memory to (temporarily) store `R`.

To prevent running out of memory even for relatively small n, we not only clear the variables right after use, but also break up the work by dividing it into smaller steps: instead of working on the whole $n \times n$ random matrix, we divide that matrix into four blocks of about equal size and only look one by one at the top left, top right and bottom right block. Having "filtered" the

[1] That is, without the main diagonal to prevent self–loops

indices of where to place edges, we create them, Line 22, and then finally add the transpose.

The reason for using the memory consuming method of first generating these sometimes rather big random matrices is, that it appears to be significantly faster than using nested loops, randomly deciding for each entry, one by one, if there should be an edge or not.[2]

```
1  function [ A ] = gen_er( n , p )
2
3  % distinction between odd and even n ...
4  if mod(n,2), n1 = floor(n/2); n2 = n1 + 1; % odd n
5  else, n1 = n/2; n2 = n1; end        % even n
6
7  % top left block
8  R = rand(n1,n1);                    % generate rand matrix
9  T = sparse(triu(R,1)); clear R;     % only keep upper triang part
10 [ii1,jj1] = find(T>(1-p)); clear T; % determine "winning" edges
11
12 % top right block
13 R = rand(n1,n2); [ii2,jj2] = find(R>(1-p)); clear R;
14
15 % bottom right block
16 R = rand(n2,n2); T = sparse(triu(R,1)); clear R;
17 [ii3,jj3] = find(T>(1-p)); clear T;
18
19 % combine indices
20 ii = [ii1 ; ii2; ii3+n1]; jj = [jj1 ; jj2+n1 ; jj3+n1];
21 % create A and add transposed to get the full, symm. adj. matrix
22 A = sparse(ii,jj,true(1,length(ii)),n,n); A = A | A';
```

Listing 2.1: *Generating program for Erdös–Rényi graphs.*

2.2.2 Properties

In their original article Erdös and Rényi defined the random graph as a graph with n nodes and e edges chosen randomly from the $n(n-1)/2$ possible edges. This corresponds to picking (with equal probability) one graph out of the $\mathrm{C}_e^{[n(n-1)/2]}$ possible graphs with n nodes and e edges, which form the probability space.

Both definitions are equivalent, and if we start with n nodes and connected independently every distinct pair of nodes with probability p, then the total number of edges will be a random variable. It is easy to see that its expectation value will be $\mathrm{E}(e) = p \cdot n(n-1)/2$.

Studying the properties of the probability space associated with this type of random graph on n nodes, as n tends to infinity, one of the most important observations of Erdös and Rényi was, that many important properties of this type of random graph appeared quite suddenly. Before commenting

[2] Using a 2.8 GHz Pentium 4 machine with 1024 MB of RAM, the algorithm works fine and in the order of seconds for networks of sizes up to 10000 nodes if p is not *too* large, which is more than enough for our purposes.

on this however, we would like to present the list of characteristic values for Erdös–Rényi graphs.

Characteristic values

As mentioned above, we limit ourselves to presenting the results, but give references for further reading.

Degree distribution It is relatively straightforward to show[3] that the distribution is binomial:

$$p(k) = C_{n-1}^{k} p^k (1-p)^{n-1-k} \tag{2.1}$$

For large n this distribution can be approximated by a Poisson distribution

$$p(k) \simeq \mathrm{e}^{\langle k \rangle} \frac{\langle k \rangle^k}{k!}$$

A typical degree distribution is shown in Figure 2.2(b).

Average node degree Using the expectation value from Subsection 2.2.2:

$$\langle k \rangle = 2e/n = p \cdot (n-1) \simeq pn$$

Average path length Can be found in [23]:

$$\langle l \rangle = \frac{\ln n - \gamma}{\ln(pn)} + \frac{1}{2}$$

where γ is the Euler–Mascheroni constant, see List of Symbols

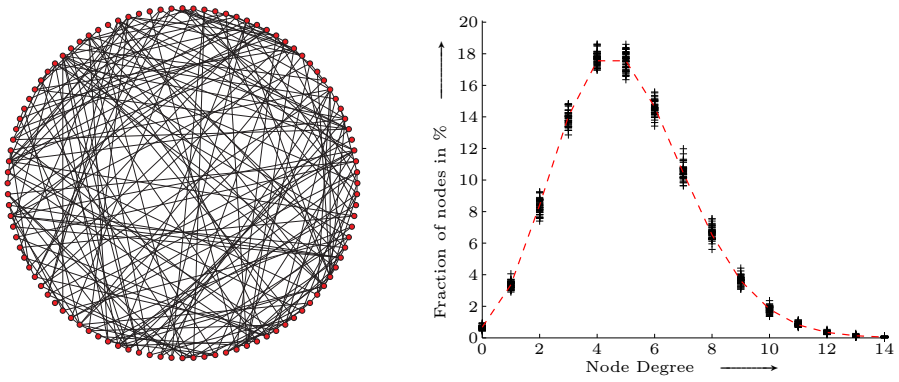
Diameter More details in [15], but usually concentrated on a few values around

$$d \approx \frac{\ln n}{\ln(pn)} = \frac{\ln n}{\ln\langle k \rangle}$$

Clustering coefficient For each node, the probability that its neighbours are connected equals p, so it follows immediately that

$$\langle c \rangle = p = \frac{\langle k \rangle}{n}$$

[3] It is made up of three parts: the number of possibilities of choosing k nodes out of $(n-1)$ other nodes (as link–targets), the probability of a particular node actually having k edges attached to it as well as the probability of *not* having $(n-1-k)$ edges attached to it.



(a) *(b)*

Figure 2.2: ***(a)*** *A random graph with $n = 100$ and $e = 200$, generated using $p = 0.04$.*
(b) *Degree distributions of 50 random graphs generated with the Erdös–Rényi model using $n = 3000$ and $p = 0.001$. The dashed line has been plotted using the binomial distribution Equation (2.1) on the facing page.*

Thresholds

Definition As mentioned above, Erdös and Rényi discovered the usually very sudden onset of many properties[4] $\mathfrak{P}$ there usually seems to be a quite precise threshold value p_c for the connection probability so that either almost no graph constructed with a $p < p_c$ has that property, or, conversely, almost every graph with $p > p_c$ has it.

For most of these properties, the critical probability is a function of n, say $p_c(n)$. So, loosely speaking, if in an Erdös–Rényi graph the edge connecting probability $p(n)$ grows faster than $p_c(n)$, i. e. we have $p(n) > p_c(n)$ as n grows, then almost every graph on n nodes with that connection probability $p(n)$ will have that property.

This can formally be written down by saying for all graphs on n nodes "built" using a connection probability of $p(n)$, the probability $p_n(\mathfrak{P})$ of these graphs having property $\mathfrak{P}$ (with critical probability $p_c(n)$) is

$$\lim_{n\to\infty} p_n(\mathfrak{P}) = \begin{cases} 0 & \text{if} \quad \dfrac{p(n)}{p_c(n)} \to 0 \\ \\ 1 & \text{if} \quad \dfrac{p(n)}{p_c(n)} \to \infty \end{cases}$$

Let's have a look at some examples.

Connectedness An interesting threshold probability would be the one for which almost every graph will be connected. It is mentioned in [2] that this

[4] For example connectedness of the graph, emergence of certain types of subgraphs, ...

is usually the case if $\langle k \rangle \geq \ln n$, resulting in $p_c \approx \ln n/n$, but a more rigorous approach gives the following theorem, found in [11]:

Theorem 2.1 (Connectedness of Erdös–Rényi graphs)

Let $c \in \mathbb{R}$ be fixed and let $\mathscr{G}_{n,p(n)}$ be an Erdös–Rényi random graph on n nodes created using

$$p(n) = \frac{\ln(n) + c + o(1)}{n} \tag{2.2}$$

Then the probability p_{conn} that $\mathscr{G}_{n,p(n)}$ is connected tends to

$$p_{conn}\left(\mathscr{G}_{n,p(n)}\right) \xrightarrow{n\to\infty} \mathrm{e}^{-\mathrm{e}^{-c}} \tag{2.3}$$

where e *stands for Euler's number,* $\mathrm{e} = \exp(1)$.

Proof: Also given in [11]. □

Subgraphs Another set of connection probability thresholds can be found in [2]. There we find, for example, that if we build Erdös–Rényi graphs with n nodes and scale the connection probability used according to $p(n) = cn^{-k/l}$ with a sensibly chosen $c \in \mathbb{R}$, many critical probabilities can be established (proofs of which, again, for example in [11]).

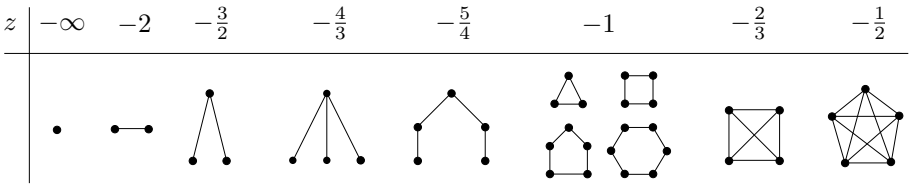


Figure 2.3: *Threshold probabilities for the appearance of different types of subgraphs for $p_c(n) \sim n^z$. Courtesy of [2].*

Just to name a few: in order for almost every graph to contain at least one

- subgraph with k nodes and l edges: $p_c(n) = cn^{-k/l}$
- tree of order k: $p_c(n) = cn^{-k/(k-1)}$
- cycle of length k: $p_c(n) = cn^{-1}$
- complete subgraph on k nodes: $p_c(n) = cn^{-2/(k-1)}$

Some of these are shown in Figure 2.3.

Giant cluster It is intuitive that, for low probabilities, Erdös–Rényi graphs consist of just a few, isolated edges, sparsely distributed throughout the graph. Increasing p, one witnesses the emergence of tress, cycles and other types of subgraphs, but which are still small and isolated clusters.

For $p(n) = cn^{-1}$, it follows immediately that the average node degree $\langle k \rangle$ is constant. While for $c < 1$, the graph consists of isolated clusters, a giant cluster

(clearly distinguishable by it's size, containing at least about $n^{2/3}$ nodes) starts to form for values of $c \geq 1$.

This phenomenon — passing from a fragmented system to a graph which is dominated by a single giant cluster, the transition roughly happening at $p_c(n) \simeq 1/n$ — is similar to a transition in infinite–dimensional percolation, a topic thoroughly studied, not only in mathematics [27].

With these remarks we would like to move on to a more recent random graph model, namely the Watts–Strogatz model.

2.3 The Watts–Strogatz or "small–world" Model

In the late 1960s, the famous American psychologist Stanley Milgram proclaimed his thesis about the "six degrees of separation" between any two persons in the USA, meaning, that any two randomly chosen individuals are linked by a chain of six or fewer first–name acquaintances [44].

Social networks indeed have been shown to feature — despite their large network size and sparse connections — short average path lengths (which coined the term "small–world"), but are still highly clustered. For instance, everybody has a local circle of friends, acquaintances through family, education, work or hobbies. Furthermore, we also have a few "long range connections" through relatives that moved abroad, an extended stay in an overseas country or somebody from china you got to know in your favoured café.

To see both these features of "clusteredness" *and* short average path lengths in Erdös–Rényi graphs, cf. Subsection 2.2.2 on page 20, one would need massive amounts of edges, as there is little order among them. In the real world however, only relatively few edges are needed for both these characteristics. In 1999, Watts and Strogatz defined this small–world concept [62], showed that many real networks have small–world characteristics and introduced an algorithm for their creation.

Before describing this algorithm and studying the characteristics of the graphs generated with it, it is important to mention that the small–world *character* can also be found in other types of networks, for example in scale–free graphs which we will present in Section 2.4 on page 28.

A typical small–world graph is shown in Figure 2.5(a).

2.3.1 Generation of small–world graphs

The idea The idea behind generating graphs with small–world character is quite simple and, on second though, also quite intuitive: start with order and randomize a bit. Starting off with order allows for the relatively high clustering, yet randomizing by rewiring some edges creates "long distance shortcuts" which are responsible for the relatively low average path length.

Watts and Strogatz chose a one dimensional lattice with a periodic bounding condition as starting point: take n nodes, lay them out to form a "ring" and connect each node with its k_0 neighbours. In order for that lattice to be symmetrical, k_0 must be even (for example two nodes to the "left" and two nodes to the "right" would mean $k_0 = 4$). Once this regular lattice is generated,

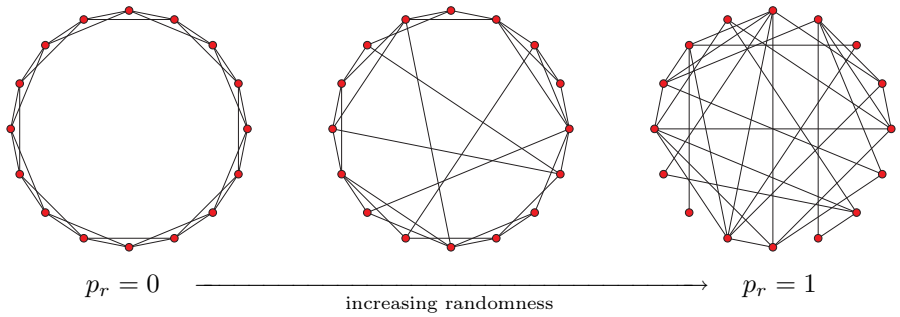


Figure 2.4: *The Watts–Strogatz model, transition from order to randomness through rewiring, in this graph with $n = 16$ and $k_0 = 4$ (so $e = 32$).*

edges are randomly rewired — random both in terms of whether or not to be rewired in the first place, and, if so, where to connect the free end to.

This process is depicted in Figure 2.4: the rewiring probability p_r passes from 0, corresponding to the initial regular 1D ring lattice (here with $k_0 = 4$) via a phase between order and complete randomness (where "local" clusters are connected via a few long range connections) to complete randomness ($p_r = 1$), corresponding to an Erdös–Rényi graph on n nodes and $e = n \cdot k_0/2$ edges.

The code With this process in mind, we can now take a look at the implementation shown in Listing 2.2 on the next page. Like most of the other functions in this document, this one is also designed to save some work by supposing undirected graphs and only working on the upper triangular part (adding the transposed in the last step).

A few comments on techniques involved: after creation of the regular lattice in Lines 5–8, the algorithm looks at each edge rolling its virtual die to "decide" whether to rewire that particular edge. If we take only the upper triangular part of the adjacency matrix, rewiring an edge corresponds to "shifting" the corresponding 1–entry to somewhere else within the same row.

To implement that, we fetch a random permutation of $\{1, 2, \ldots, n\}$ and each element as a target for where to reconnect the edge to. Conditions for that are that the spot is neither occupied (Line 21) nor on the main diagonal (Line 22). The first such spot to be found is then used to create an edge there (Lines 28–32), and the old edge gets "deleted" (Line 35).

2.3.2 Properties

Before commenting on the "emergence" of small–world behaviour, that is located somewhere between the two extremes of $p_r = 0$ and $p_r = 1$, we would like to present the "list" of characteristic values for this type of graph.

Characteristic Values Again, proofs can be found in the references:

```
function [ A ] = gen_sw( n , k_init , p )

% step 1: generate regular 1D-lattice
A = logical(sparse(n,n));
for k=1:k_init/2
   A = A | spdiags(ones(n,1),k,n,n);
   A = A | spdiags(ones(n,1),(n-k),n,n);
end

% step 2: rewire: look at each edge and 'decide' whether to rewire
[i,j] = find(A); rewire_counter = 0;

for curr=1:length(i)

   if rand ≤ p % go ahead, rewire !
      rewire_counter = rewire_counter+1;

      % now find a 'spot' for the new egde...
      l = 1; j_attempt = randperm(n);% rand. perm. of col. ind.
      %              ...which is neither occupied...
      while A(i(curr),j_attempt(l))||A(j_attempt(l),i(curr)) ...
            || ( j_attempt(l) == i(curr) % ...nor on main diag
         l = l+1;
      end

      % found candidate -> perform the actual rewiring:
      % create new edge in upper trinang. part
      if i(curr) > j_attempt(l) % would end up in lower triang.
         A(j_attempt(l),i(curr)) = true;
      else
         A(i(curr),j_attempt(l)) = true;
      end

      % and remove old edge
      A(i(curr),j(curr)) = false;
   end
end
% finally add transposed to get the full, symm. adj. matrix
A = A | A';
```

Listing 2.2: *Generating program for small–world graphs using the Watts–Strogatz model.*

Degree distribution Found in [6]:

$$p(k) = \sum_{n=0}^{\min\{k-\tilde{k}_0,\tilde{k}_0\}} \mathrm{C}_{\tilde{k}_0}^{n} (1-p_r)^n p_r^{\tilde{k}_0-n} \frac{(p_r\tilde{k}_0)^a}{a!} \exp\left(-p_r\tilde{k}_0\right) \tag{2.4}$$

for $k \geq k_0/2$, with $\tilde{k}_0 = k_0/2$ and $a = k - \tilde{k}_0 - n$. A typical degree distribution is shown in Figure 2.5(b).

Average node degree As edges are only rewired, obviously

$$\langle k \rangle = k_0$$

Average path length To the best of our knowledge, an exact solution has not be found yet. In [49] however, we find a good approximation for a model very similar to the one discussed here. The only difference is that instead of rewiring edges, shortcuts are added directly (thus leaving the original ring lattice "intact", but increasing the total number of edges).[5] There,

$$\langle l \rangle \simeq \frac{\xi}{k_0\sqrt{1+2\xi/n}} \tanh^{-1}\left(\frac{1}{\sqrt{1+2\xi/n}}\right) \tag{2.5}$$

where $\xi = 2/(k_0 p_s)$ with p_s is the probability similar to the Erdös–Rényi model of having a shortcut between two nodes.

Diameter Again, to the best of our knowledge no firm analytic results are available at the time of writing.

Clustering coefficient A slightly different but equivalent definition of the clustering coefficient than that given in Subsection 1.4.4 would be to define it as the fraction of the mean number of edges between the neighbours of a node and the mean number of possible edges between those neighbours.

For this definition of clustering, we find in [6]:

$$\langle c \rangle' = \langle c \rangle_0' (1-p)^3 = \frac{3(k_0/2-1)}{2(k_0-1)}(1-p)^3$$

and the deviation from $\langle c \rangle$ is of order $1/n$. So we can readily say

$$\langle c \rangle \simeq \langle c \rangle_0 (1-p)^3 \tag{2.6}$$

the index 0 denoting, again, the value for $p_r = 0$.[6]

Influence of the rewiring probability As we have mentioned earlier on, typical small–world graphs show a remarkable combination of high clustering (which is a local property) and short average path lengths (a global measure).

In order to stick closely to properties of some real networks (like social networks), Watts and Strogatz were interested in graphs with many vertices but few connections between them (but not so few that the graph would be in danger of becoming disconnected). They specified their request by demanding

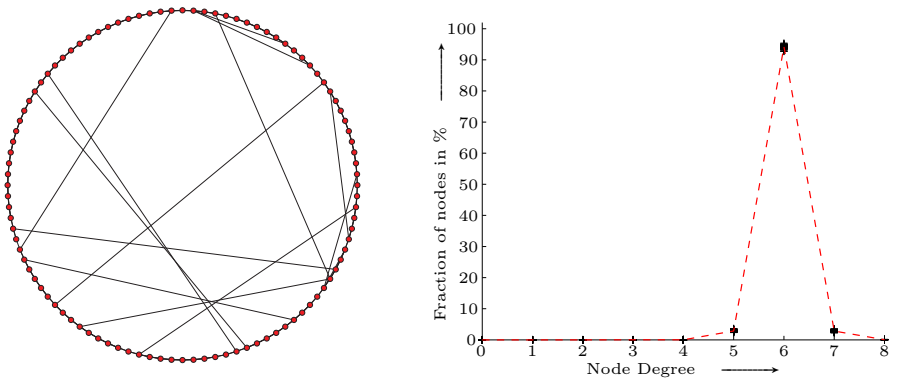
$$n \gg k_0 \gg \ln n \gg 1$$

Here, $k \gg \ln n$ guarantees that the resulting graph will be connected, [10]. With their model (and appropriately chosen parameters fulfilling this request) it is easy to obtain the desired high clustering and short distances. Again, an Erdös–Rényi graph with similar properties would need to have significantly more edges.

However, even with the Watts–Strogatz model the desired properties are not always present. In fact, the emergence of this behaviour depends strongly on p_r. If we look at both extremes, we find

[5] Both models are equivalent for sufficiently small p_r respectively p_s and large n.

[6] It can easily be established that $\langle c \rangle_0 = \frac{3(k_0-2)}{4(k_0-1)}$.



(a) *(b)*

Figure 2.5: ***(a)*** *A small–world graph with $n = 100$ and $e = 200$, generated with the Watts–Strogatz model using $p_r = 0.05$ and $k_0 = 4$.*
(b) *Degree distributions of 50 small–world graphs generated using $n = 3000$, $k_0 = 6$ and $p_r = 0.01$. The dashed line has been plotted using the distribution Equation (2.4) on page 25.*

$$\langle c\rangle_0 = \frac{3(k_0-2)}{4(k_0-1)} \simeq \frac{3}{4} \longrightarrow \langle c\rangle_1 \sim \frac{k_0}{n}$$

$$\langle l\rangle_0 = \frac{n(n+k_0-2)}{2k_0(n-1)} \simeq \frac{n}{2k_0} \xrightarrow{0\to p_r\to 1} \langle l\rangle_1 \sim \frac{\ln n}{\ln(k_0-1)}$$

thus for small p_r, $\langle c\rangle$ seems to be large and $\langle l\rangle$ scales linearly with n, whereas for large p_r the clustering coefficient seems to decrease with the system size and the average path length only scales logarithmically with n.

Both these extremes and intuition seem to suggest that a large $\langle c\rangle$ is always associated with a large $\langle l\rangle$, and small clustering with short path lengths.

Figure 2.6 on the following page however shows that $\langle c\rangle$ does not decrease as fast as $\langle l\rangle$, leaving a broad range of values p_r resulting in networks that are highly clustered, but still with small characteristic path length — Watts and Strogatz' small–world networks.

The prediction for the clustering coefficient fits perfectly the numerical results; however the theoretical results for the average path length stay below the simulated data. This is surely due to the fact that the slightly different model they are valid for *adds* shortcuts instead of rewiring edges. The resulting increased number of edges allows for shorter average path lengths than in the case where edges are only rewired.

We shall now continue with our third and last random graph model, the Barabási–Albert model.

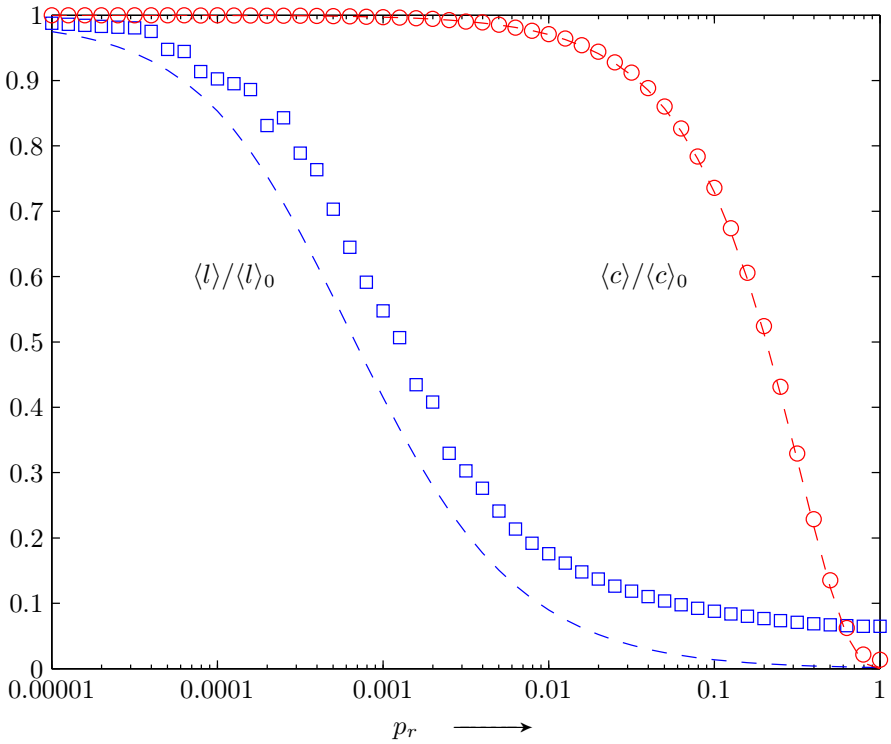


Figure 2.6: *Normalized clustering coefficient $\langle c\rangle$ (○) and average path length $\langle l\rangle$ (□), as a function of p_r, normalized with the corresponding values for the regular lattice i. e. $p_r = 0$).*
Each data point corresponds to the average value over 50 independently generated small–world graphs on $n = 1000$ nodes with $k_0 = 10$ neighbours per node, normalized by the corresponding value for $p_r = 0$.
The dashed lines correspond to the predictions of Equation (2.5) respectively Equation (2.6) on page 26.

2.4 The Barabási–Albert or "scale–free" Model

As we will see in Subsection 2.5.2 on page 35, many if not most of the large real world networks appear to have a power law degree distribution

$$p(k) \sim k^{-\gamma} \tag{2.7}$$

with the exponent γ usually somewhere between 1 and 4. This distribution deviates significantly from those characteristic to the Erdös–Rényi and Watts–Strogatz models, especially in not having a typical or "meaningful" average value and in allowing for a few but very highly connected nodes. In addition, the probability for a certain node to have a certain degree does not depend on the size of the network. These facts are responsible for the term "absence of scale".

One model to account for similar degree distributions and characteristics is based on the assumption that these properties must be due in part to the

evolution inherent to certain types of networks: on one hand they usually show some sort of growth, and, on the other hand, new nodes introduced into the network usually connect preferentially to the more highly connected nodes already present in the network. This non uniform connection probability seems to introduce certain correlations and dependences that allow for some of the specific attributes of these graphs.

In addition, as the power law degree distribution allows for some (very few but) extremely high node degrees, many real world networks have a certain maximum node degree, or cutoff, that limits or bounds the range of node degrees. Here, the plot of the degree distribution deviates from a straight line in a double logarithmic plot to show an exponential or "Gaussian" tail. Amaral *et al.* give a more detailed introduction to these phenomena, [3].

Building a suitable model for these types of networks has been an area of high interest in the past few years and a multitude of models has since arisen. Table III in [2] gives an excellent overview of the overwhelming number of variations on the model. In this paper however, we will limit ourselves to the seminal Barabási–Albert model, which gave rise to the burst of activity in the field. A typical scale–free graph generated with their model is shown in Figure 2.8(a).

2.4.1 Generation of scale–free graphs

The idea Although it is straightforward to generate networks with a specific degree distribution (just by generating node degrees according to the specific distribution and then randomly placing the edges accordingly), until lately, those networks failed to show key properties observed in real world networks with a power law distribution.

One reason for this was that the usual generating method did not reflect the dynamical process that creates real world networks, introducing nontrivial correlations that affect many topological properties. Seeing the evolution of real world networks (like the world wide web, which started with a couple of hundred pages, and now has grown to many billion pages), the generating algorithm for an artificial network should also incorporate a similar growth process as well as the preferential attachment (as, for example, some new page is much more likely to link to an important page like www.google.com than to *some* rather insignificant page).

Both ideas inspired the first model introduced by Barabási *et al.* in 1999, [4], which generated a scale–free network with a power law degree distribution *and* many of the properties encountered in real networks. This model allowed for the first time to describe the inherent ordering principle observed in real world networks. Its simple but elegant generating algorithm is the following:

(i) *Growth*: Start with a very small but fully connected graph on n_0 nodes ($1 \leq n_0 \ll n$, n being the desired final number of nodes). At each iteration step t, add one node with m edges that link the new node to m nodes that already are in the system.

(ii) *Preferential attachment*: The "targets" for the m new edges to be placed are not chosen with equal probability, but with probability proportional

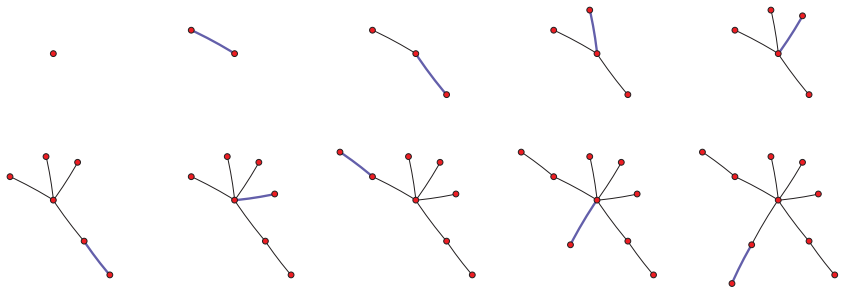


Figure 2.7: *Growth of a scale–free graph using the Barabási–Albert model, starting with $n_0 = 1$ node and adding one node and $m = 1$ edge at each step (these new edges are marked with ——).*

to their respective degrees:

$$p_{a_i}^{(t)}\left(k_i^{(t)}\right) = \frac{k_i^{(t)}}{\sum_{j=1}^{n_0+t-1} k_j^{(t)}} \tag{2.8}$$

where $p_{a_i}^{(t)}$ denotes the probability that the new node attaches an edge to node i, which has degree $k_i^{(t)}$ at step t.

So in order to get a network with n nodes, this algorithm needs to be run for $t = n - n_0$ steps, resulting in a total of $(n - n_0) \cdot m$ edges.

The attentive reader should immediately ask: what happens at $t = 1$ for $n_0 = 1$, when the sum in the denominator of Equation (2.8) is zero? Well, in this particular case we need some "faking", setting $p_{a_1}^{(0)}$ to 1 to start off the algorithm.

Figure 2.7 shows an example of the first nine steps of generating a scale–free graph using the Barabási–Albert model described above, starting off with $n_0 = 1$ node and adding $m = 1$ edge with each newly introduced node.

The code Our implementation of this algorithm, shown in Listing 2.3 on page 32, uses a "Monte Carlo" technique to actually place the edges according to the particular distribution induced by the preferential attachment Equation (2.8).

For that, we calculate the *cumulative* probabilities in Line 26. There, the `pcum` vector looks like $\left(p_{a1}(k_1) \quad , \quad p_{a1}(k_1) + p_{a2}(k_2) \quad , \quad \ldots \quad , \quad \sum_i p_{ai}(k_i)\right)$ some step iteration step. This vector calculated, we can generate a uniformly distributed random number between zero and one and then see in which "area" of the `pcum` vector it "falls": larger p_c's produce broader intervals in `pcum` and are thus more likely to "catch" the random number, i. e. get the edge. This is done in Line 36, where the first element in the `temp` corresponds to the node number that produced the interval the random number fell in.

We placed a `while`–loop around this procedure that keeps repeating this process until an "empty" spot for the new edge has been found (the more "attractive" a location is the more likely it will be that an edge has been placed there already in a previous step), Line 33.

Finally, in order to create a network with n nodes, $n - n_0$ repetitions are needed (Line 15), within each of which m edge–placement–runs need to be carried out (Line 30).

2.4.2 Properties

Now that we are able to generate the basic type of scale–free networks, let's take a closer look at their properties, starting off with the characteristic values.

Characteristic Values Again, proofs can be found in the references:

Degree distribution Found in [2]

$$p(k) = \frac{2m(m+1)}{k(k+1)(k+2)} \sim k^{-3} \tag{2.9}$$

A typical degree distribution is shown in Figure 2.8(b).

Average node degree Straightforward:

$$\langle k \rangle = 2e/n = \frac{2m(n-n_o)}{n} \simeq 2m$$

Average path length In [23] it is shown that in a good approximation

$$\langle l \rangle \simeq \frac{\ln n - \ln(m/2) - 1 - \gamma}{\ln(\ln n) + \ln(m/2)} + \frac{3}{2} \tag{2.10}$$

Diameter Using a similar model, with the difference of allowing for multiple edges and self loops, we find in [12] for very large n and $m \geq 2$ that d is usually concentrated on a few values around

$$d \approx \frac{\ln n}{\ln(\ln n)}$$

For $m = 1$ it is shown in the same paper that d scales as $\ln n$.

Clustering coefficient We find in [22]:

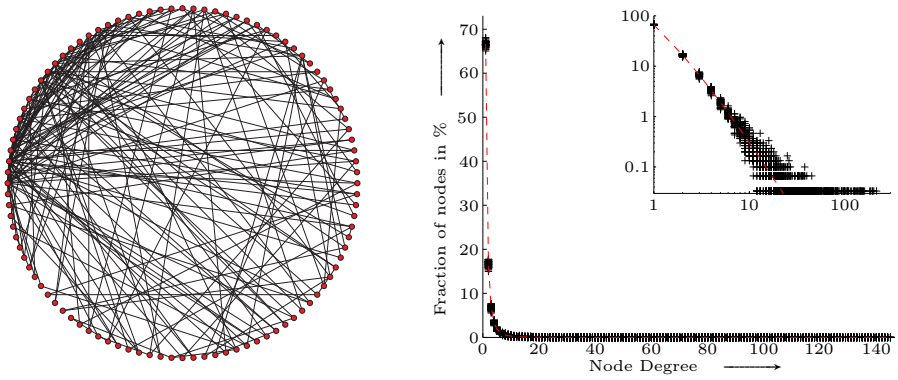
$$\langle c \rangle = \frac{6m^2\left[(m+1)^2(\ln t)^2 - 8m\ln t + 8m\right]}{8(m-1)(6m^2+8m+3)t}$$

Ultra small–world If we take a look at Equation (2.10), we can see that for large n the average path length scales as $\ln n/\ln(\ln n)$, which increases much more slowly than $\ln n$, a suggested asymptotic form for graphs generated by both the Watts–Strogatz and Erdös–Rényi model, see for example [49].

Besides an alternative way of finding the approximation $\langle l \rangle \sim \ln n/\ln(\ln n)$, it is claimed in [16] that scale–free networks are "ultrasmall". In Figure 2.9 on page 34 we compare the growth of $\langle l \rangle$ as n increases. To make the graphs comparable, the respective generating parameters have been chosen in a way that for each graph $e \simeq 4n$. This relatively high number of edges is needed

```
function [ A ] = gen_sf( n , n_0 , m  , quiet)

if n_0 == 1       % danger of dividing by zero (k=0...)
   tweak = true;
else              % everything is fine
   tweak = false;
end

% create empty A
A = logical(sparse(n,n));

% create initial (fully connected) cluster:
A(1:n_0,1:n_0) = true(n_0,n_0) - speye(n_0,n_0);

for t = 1:n-n_0   % time-step-loop

   if tweak
      pcum = 1;   % faked cumulative sum for n_0=0 @ t=1
      if t > 1    % out of trouble -> stop faking
         tweak = false;
         % there are 2 nodes and 1 edge -> p_a = ( 1/2 , 1/2 )
         pcum = [.5 1];
      end
   else           % not tweaking - use p_a_i = deg_i / sum(deg)
      deg = sum( A(1:n_0+t-1,1:n_0+t-1) , 2); % node degrees
      pcum = cumsum( deg / sum(deg) ); % cum probs
   end

   % now, let's place the m new edges
   for j = 1:m

      succeeded = false;
      while ¬succeeded % while not succeeded placing new edge...

         % determine target for new edge
         temp = find(pcum>rand);

         % check if we can place the edge here
         if ¬A(temp(1),n_0+t)
            succeeded = true;
         end
      end % placement-loop

      % now, place the edge
      A(temp(1),n_0+t) = true; A(n_0+t,temp(1)) = true;

   end % edge-loop

end % node-loop
```

Listing 2.3: *Generating program for scale–free networks using the Barabási–Albert algorithm.*



(a) *(b)*

Figure 2.8: ***(a)*** *A scale–free graph with $n = 100$ and $e = 196$, generated with the Barabási–Albert model using $n_0 = 2$ and $m = 2$.*
(b) *Degree distributions of 50 scale–free graphs generated using $n = 3000$, $n_0 = 1$ and $m = 1$. The dashed line has been plotted using the distribution Equation (2.9) on page 31. The increasing "spreading" for higher node degrees in the double logarithmic inset is due to the discrete nature and the resulting limited "resolution" of the experiment.*

to "ensure" connected Erdös–Rényi graphs (to be able to compare the average path lengths).

We can see that the average path length for scale–free graphs grows much slower than for the small–world graphs and slightly slower than for Erdös–Rényi graphs. The "separation" between the latter two however increases with growing n.

The following section is dedicated to a more extensive comparison of the three models we have now introduced.

2.5 Comparison and applications

2.5.1 Characteristic values

To start this section, we present a lineup of the three models with respect to their characteristic values. This is done empirically by calculating these values using the algorithms described in the first chapter for the three models generated with the algorithms described above.

Table 2.1 on page 35 shows next to each calculated value — which is the average value of 200 independent repetitions per graph model and characteristic value — the theoretical value given by the lineup of equations we presented for each model.

The graphs used each have $n = 2000$ nodes and a targeted $e \simeq 4000$ nodes. We generated them using the following parameters:

– Erdös–Rényi: $n = 2000$, $p = 4/1998$

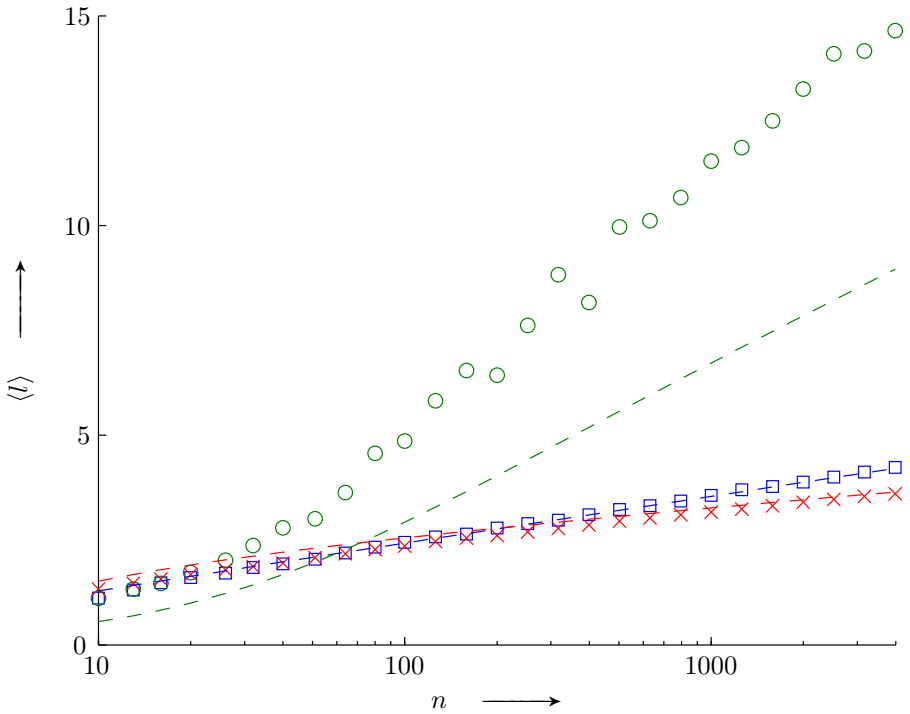


Figure 2.9: *Evolution of average path lengths as network size (and number of edges $e \simeq 4n$) increases. The graph models used are: Erdös–Rényi with $p = 8/(n-1)$ (□), Watts–Strogatz with $k_0 = 8$ and $p_r = 0.01$ (○) and Barabási–Albert with $n_0 = m = 4$ (×).*
Each datapoint corresponds to the average value from 5 independent runs, the dashed lines show the respective predicted values.

- Watts–Strogatz: $n = 2000$, $k_0 = 4$, $p_r = 0.016$ [7]
- Barabási–Albert: $n = 2000$, $n_0 = 2$, $m = 2$

The results from these runs give us some numerical evidence for and some idea of validity of the specific features and predicted values mentioned above:

Average node degree These are, as expected, roughly the same as the networks are targeted to have twice the number of edges than nodes.

Average path length We can see that the scale–free networks as well as the Erdös–Rényi type networks perform much better than Watts–Strogatz networks when it comes to the average distance between two nodes. We can also see the "ultrasmall" character of the scale–free graphs as they have even shorter path lengths than the Erdös–Rényi ones. The predicted values are also quite in agreement with the numerical results (but for deviation in the Watts–Strogatz model), as we could also see in Figure 2.9.

[7] This specific value for p_r has been chosen using a plot similar to Figure 2.6 on page 28 to create maximum clustering but at the same time shortest average path lengths.

Diameter These values are unfortunately relatively different from their predictions, or even undetermined as in the Watts–Strogatz case. Concerning the deviations, however, the values are still in accordance with the bounds surrounding them (see [15] and [12] for further details).

Clustering coefficient As expected, clustering is much higher in small–world networks than in the others. As we mentioned earlier in the introduction to the Watts–Strogatz model, the scale–free graphs also shows some small–world behaviour, as their clustering coefficient is still about 8 times higher than that of a random graph. For every graph model, prediction and simulation agree very well.

Value	Erdös–Rényi		Watts–Strogatz		Barabási–Albert	
	num.	theor.	num.	theor.	num.	theor.
$\langle k \rangle$	4.001	4.000	4.000	4.000	3.996	3.996
$\langle l \rangle$	5.599	5.565	25.549	18.723	4.373	4.470
d	11.850	5.481	62.570	?	7.990	3.748
$\langle c \rangle$	0.002	0.002	0.477	0.476	0.015	0.015

Table 2.1: *Comparison of the characteristic values between the three network models as well as between the calculated and theoretical values. Each of the "numerical" values are the average values gained from 200 independently generated networks having $n = 2000$ nodes and a targeted $e = 4000$ edges.*

2.5.2 The real world

To close this chapter on random graphs we would like to present some examples of real networks whose characteristics can be particularly well reflected by some of the models presented above.

World wide web and internet The world wide web (as of the network of web pages connected by hyperlinks) and also the internet (as of the network of interconnected computers, routers, servers, etc.) are the largest networks with quite precisely measurable topological data available today.

In contrast to the internet however, the world wide web is usually approximated by directed graphs, resulting in two degree distributions for web pages — one for in– and another one for the out–degrees of pages. It has been shown in several papers (to name a few [14, 26, 60]) that the internet (on router and domain level) as well as the world wide web (at domain and page level) has node degrees displaying a clear power law distribution with γ for in– and out–degrees usually between 2.0 and 2.5.

Phone call networks Another intuitive network that can be established is that of phone call patterns, where phone numbers are the nodes, connected by (directed) arcs representing calls. Studies of long distance calls on a single

day [1] have shown that such graph also has a power law in– and out–degree distribution with exponent $\gamma \simeq 2.1$.

Citation networks A scale free degree distribution can also be found in graphs describing citations, nodes being documents and directed edges citations between them. Highly connected nodes would correspond to some few seminal papers where as larger numbers of less significant papers have only few connections. A study [54] showed that the probability of a paper being cited k times can be described with a power law, using $\gamma \simeq 3$.

Human sexual contacts Based on a Swedish survey from 1996 [40], an interesting study [41] shows that the number of sexual partners up to 12 months prior to the survey is distributed according to a power law, with exponent of about $\gamma \simeq 2.4$. Besides that, the data and heuristic explanations show that these networks are consistent with the preferential attachment mechanism discussed above.

Collaboration networks Large and publicly available databases like IMDB[8] or arXiv.org[9] allow close studies of another set of networks that can be established, namely those capturing collaborations. Here, actors or scientists make up the nodes, and are linked together if they have starred together in a movie respectively were co–authors of some paper.

Several papers show studies of these networks, [5, 50, 62], and find power law exponents usually around $\gamma = 2.4$.

Neural networks and power grids Neural networks like those of small worms or power grids like those spanning the western United States [3] show similar properties, namely that the average path lengths are close to those of comparable Erdös–Rényi graphs, but clustering is significantly higher.

In [62], when Watts *et al.* introduced the small–world model, they showed that the properties of these networks as well as the previous two social networks can be reflected very well by their model.

Protein folding In general, protein folding can be seen as the process a protein goes through in order to assume its functional shape or conformation. As protein molecules are chains of simple, unbranched amino acids, it is through coiling into a certain three–dimensional shape that they perform their specific biological function.

During that process, the different (consecutive) configurations a protein goes through can be described with a graph, where each node corresponds to a particular conformation, two nodes being linked together if they can be obtained from each other by an elementary move. Several papers show that these networks show significant small–world characteristics, [53, 55, 61] and typically have scale–free degree distributions.

[8] That is the Internet Movie Database (http://www.imdb.com)

[9] The e–Print archive arXiv.org (http://www.arxiv.org) actually provides free and publicly accessible copies of the majority of the papers and articles cited in this document.

This biologically motivated topic will actually be treated in more detail in Chapter 5. We shall now continue, however, with a number of ranking schemes that we will use to identify important nodes in a network.

CHAPTER 3

Ranking schemes

We present methods to rank nodes, including node degrees, algorithms like HITS and PageRank, centrality based measures like excentricity, status or centroid value, and "damage".

3.1 Introduction

With the necessary terminology that we have gathered in the first chapter we discussed three major models for random graphs. With these models it is easy to generate graphs of arbitrary size, but also modeling the real world confronts us in many cases with large if not gigantic networks. One just has to think of the snapshot the search engine Google has taken of the internet by indexing more than 8 billion pages and capturing the hyperlinks between these pages, [25]. But also metabolic networks or protein–protein interaction networks can currently have several thousand nodes, see Chapter 5.

The sheer size of these graphs makes it quite impossible to analyse them by just "looking" at them. A much more systematic approach needs to be taken.

In many applications one question almost always arises, and has gotten a lot of attention in the last ten years especially in relation with information retrieval systems like Google:

"How do I find the most 'important' nodes in a network?"

Ignoring the actual interpretation of "importance" for the moment, we could answer the above question in a detailed and elegant way by establishing a ranking of the nodes. This can be done by calculating some sort of "score" for each node in the network — desirably in a systematic and sensible way.

The actual *notion* of importance that a ranking scheme attributes to the scores depends, of course, on the actual measure of importance used, and has to ultimately prove itself in the actual practical application.

Before focusing on that, which is done in the last chapter, we shall first present a variety of classical and more recent measures of importance accompanied by MATLAB implementations of their algorithms.

3.2 Node degrees

Maybe one of the most immediate measures of importance may be the degree of a node. The idea behind this method is, that the more neighbors a node has the more influence it may have.

However, as node degrees are an intrinsically local measure, it cannot fully account for the *global* influence a node may have. In fact, node degrees only allow for a meaningful interpretation if the graph in question is a typical instance of a *known* statistical ensemble, like coming from the Erdös–Rényi, Barabási–Albert or Watts–Strogatz model.

Moreover, the node degree can be a very granulated measure. In case of a network where two thirds of the nodes have degree 1 for example there would not be too much significance in the resulting ranking.

Its calculation being very easy and fast to accomplish (simply calculate the row sums of the adjacency matrix) it can be used as a rough first approximation. However, the shortcomings mentioned above have motivated the development of more complex measures, like HITS and PageRank.

3.3 HITS

Jon Kleinberg, at the time assistant professor at Cornell University, started to develop his *Hypertext Induced Topic Search* in 1997. Today, HITS is used in the search engine TEOMA. Kleinberg's, at the time very innovative, idea was to exploit the web's hyperlink structure [35]. Each page is represented as a node in a very large directed graph, the edges corresponding to hyperlinks.

An informative, valuable, *authoritative* page is usually pointed to by a large number of other pages on the same topic, and hence has many inlinks.

Vice versa, a web page having many high quality outlinks is also a very useful resource, especially when doing some initial research on a specific topic. In that case, we could trust this page to lead us to a number of good pages on the subject.

So the concepts of *authorities* (a document having several inlinks) and *hubs* (a document having several outlinks, cf. Figure 3.1 on the next page) are used to give each page two scores — an authority and a hub score — in a mutually reinforcing way, using the thesis

> *"Good hubs point to good authorities;*
> *Good authorities are pointed to by good hubs"*

An analogy would be to look at the head of department of some faculty being a hub, knowing what all the different specialists (authorities) are working on and being able to point you to the right person for the specific question you have. Of course, the head of department would also be a specialist in some field, and other researchers (also being hubs to some extent) may point you to him. Hence the head of department can be given both an authority score *and* a hub score.

Where for more experienced users it is an advantage to have two rankings resulting from one query, these two rankings are usually combined into a single ranking which is presented at the actual front end of an information retrieval system like TEOMA.

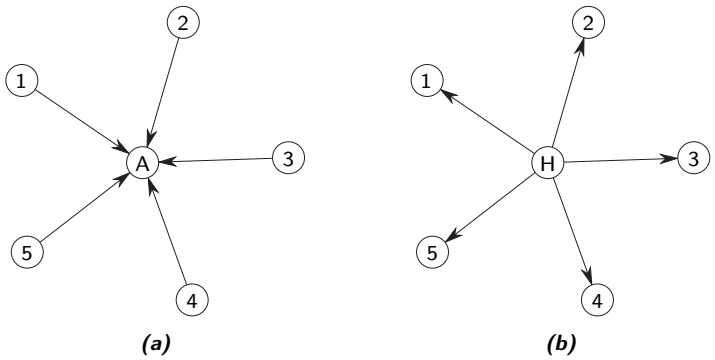


Figure 3.1: *Example for an authority* ***(a)*** *and a hub node* ***(b)****.*

Let's now take a closer look at the implementation of HITS in an internet search engine. The processing of a query usually involves two main steps:

1. Building the neighborhood graph
2. Calculating the scores

Later, however, we will only implement the second step as we already have the graphs we would like to apply the algorithm to. For reasons of completeness though we shall describe both steps, first by explaining how the neighborhood graph is obtained from a (very large) set of documents.

3.3.1 Building the neighborhood graph

First, all documents containing the query term or terms need to be identified. One simple method for this would be to look up each term in an inverted term–document file. That file is basically like a (very complete) index in a book. It might look like this:

term 1	(Aachen)	doc 4, doc 437, doc 8471, ...
	⋮	
term 9514	(Lutero)	doc 984, doc 2578, doc 12543, ...
	⋮	
term 59040	(zymurgy)	doc 64, doc 768, doc 73443, ...

For each term of the query, this file is consulted to collect all relevant document ids. Those are put into the subset $\mathcal{V}_0$ which is then used to build a directed graph $\mathcal{G}_0$. The elements of $\mathcal{V}_0$ form the nodes of that graph, whereas the arcs represent the links between them.

$\mathcal{G}_0$ is then expanded by adding all the nodes pointed to by and pointing to the documents in $\mathcal{V}_0$. Let's call this expanded subset $\mathcal{V}$. This expansion allows for some very limited latent semantic associations to be captured (for example if one query term was *car*, it is quite likely that some pages containing *automobile* are fetched as well). As with this expansion process $\mathcal{V}$ can become

very large, in practice the number of additional nodes to be added to $\mathcal{G}_0$ is usually restricted to, say, 100 per node.

Once the expansion process of $\mathcal{G}_0$ is completed (let's call the resulting graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$), the adjancency matrix $\boldsymbol{A}$ can be formed for $\mathscr{G}$. Note that the order of $\boldsymbol{A}$ is typically much smaller than the total number of documents on the web.

3.3.2 Calculating the scores

The power method

Let's quickly recall the HITS thesis: good authorities are pointed to by good hubs and good hubs point to good authorities. As said earlier, each page i in $\mathcal{V}$ gets two scores: an *authority score* x_i and a *hub score* y_i. Given some initial scores $x_i^{(0)}$ and $y_i^{(0)}$, we can then iteratively refine the scores by computing

$$x_i^{(t)} = \sum_{j:(j,i)\in\mathcal{E}} y_j^{(t-1)} \qquad \text{and} \qquad y_i^{(t)} = \sum_{j:(i,j)\in\mathcal{E}} x_j^{(t)} \qquad \text{for } t = 1, 2, 3, \ldots.$$

where $x_i^{(t)}$ and $y_i^{(t)}$ are the authority and hub scores respectively of node i at iteration step t. These two equations can be written in matrix form using the adjacency matrix $\boldsymbol{A}$:

$$\boldsymbol{x}^{(t)} = \boldsymbol{A}^T\boldsymbol{y}^{(t-1)} \qquad \text{and} \qquad \boldsymbol{y}^{(t)} = \boldsymbol{A}\boldsymbol{x}^{(t)} \qquad \text{for } t = 1, 2, 3, \ldots. \tag{3.1}$$

Note that these two equations can be simplified by substitution to

$$\boldsymbol{x}^{(t)} = \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x}^{(t-1)} \tag{3.2a}$$

$$\boldsymbol{y}^{(t)} = \boldsymbol{A}\boldsymbol{A}^T\boldsymbol{y}^{(t-1)} \tag{3.2b}$$

for $t = 1, 2, 3, \ldots$.

In fact, Equation (3.2) define a well known and investigated iterative process usually referred to as the *power method*. It is obvious that if it converges, it will do so toward eigenvectors of *authority matrix* $\boldsymbol{A}^T\boldsymbol{A}$ and *hub matrix* $\boldsymbol{A}\boldsymbol{A}^T$. More over, in this case it will converge toward the dominant eigenvectors of these matrices, as we will see in the next subsection.

So calculation of our desired authority and hub score vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ can be interpreted as finding dominant right–hand eigenvectors of the authority and hub matrices respectively. As the two of them are closely connected, we will concentrate on Equation (3.2a), as $\boldsymbol{y}$ can be easily obtained from the second equation in Equation (3.1).

Convergence

The "classic" power method, [31], assumes that we are given a diagonalizable $n \times n$ matrix $\boldsymbol{M}$ whose distinct eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_m\}$, $m \leq n$ can be ordered such that $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_m|$, and an initial vector which must not be orthogonal to the eigenspace belonging to λ_1. We then compute iteratively

$$\boldsymbol{x}^{(t)} = \boldsymbol{M}\boldsymbol{x}^{(t-1)} \tag{3.3a}$$

$$\boldsymbol{x}^{(t)} \longleftarrow \frac{\boldsymbol{x}^{(t)}}{\text{norm}(\boldsymbol{x}^{(t)})} \tag{3.3b}$$

where any norm in the $\mathbb{R}^n$ may be used. So in order to make the iteration Equation (3.3a) converge, one has to normalise $\boldsymbol{x}^{(t)}$ at each step. If Equation (3.3) converges as t tends to infinity, $\boldsymbol{x}^{(t)}$ will converge toward an eigenvector of $\boldsymbol{M}$, or in case of the HITS algorithm toward an eigenvector of $\boldsymbol{A}^T\boldsymbol{A}$.

As $\boldsymbol{A}^T\boldsymbol{A}$ is symmetric[1], positive semidefinite[2] and nonnegative[3], it is always diagonalizable and its eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ are necessarily all real and nonnegative (in other words, it is not possible to have several eigenvalues on the spectral circle). Consequently, by standard result from linear algebra [24], the HITS specific power method will *always* converge.

Uniqueness

However, it is possible that for different initial conditions different limiting vectors occur. If $\boldsymbol{A}^T\boldsymbol{A}$ or some power of it is positive, then the dominant eigenvalue λ_1 has multiplicity 1 (one of the properties guaranteed by the Perron–Frobenius Theorem, see Theorems A.2 and A.3 in the Appendix on page 98) and the power method will converge toward the unique normalized dominant (Perron–) eigenvector associated to λ_1. But if λ_1 is a *repeated* root of the characteristic polynomial, i. e. has multiplicity greater than 1, the associated eigenspace will be multidimensional and the power method would converge to some point inside that space. However, in general, the limiting vector is not uniquely determined in this case.

A simple example taken from [20] illustrates this. There,

$$\boldsymbol{A} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & 1 & \cdot \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{A}^T\boldsymbol{A} = \begin{bmatrix} 2 & \cdot & \cdot & \cdot \\ \cdot & 1 & 1 & \cdot \\ \cdot & 1 & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

The authority matrix $\boldsymbol{A}^T\boldsymbol{A}$ has two distinct eigenvalues $\lambda_1 = 2$ and $\lambda_2 = 0$, each with algebraic (and geometric) multiplicity two. For

$$\boldsymbol{x}^{(0)} = \begin{pmatrix} 1/4 & 1/4 & 1/4 & 1/4 \end{pmatrix}^T$$

the power method with 1–norm[4] normalization would converge to

$$\boldsymbol{x} = \begin{pmatrix} 1/3 & 1/3 & 1/3 & 0 \end{pmatrix}^T$$

whereas for

$$\boldsymbol{x}^{(0)} = \begin{pmatrix} 1/4 & 1/8 & 1/8 & 1/2 \end{pmatrix}^T$$

it converges to

$$\boldsymbol{x} = \begin{pmatrix} 1/2 & 1/4 & 1/4 & 0 \end{pmatrix}^T$$

---

[1] $\left(\boldsymbol{A}^T\boldsymbol{A}\right)^T = \left(\boldsymbol{A}^T\right)\left(\boldsymbol{A}^T\right)^T = \boldsymbol{A}^T\boldsymbol{A}$
[2] $\boldsymbol{x}^T(\boldsymbol{A}^T\boldsymbol{A})\boldsymbol{x} = (\boldsymbol{A}\boldsymbol{x})^T(\boldsymbol{A}\boldsymbol{x}) \geq 0$
[3] $\boldsymbol{A} \succeq \boldsymbol{0} \Rightarrow \boldsymbol{A}^T\boldsymbol{A} \succeq \boldsymbol{0}$
[4] $\|\boldsymbol{x}\|_1 = \sum_i |x_i|$

An explanation In fact, at the heart of this uniqueness problem is the reducibility of $\boldsymbol{A}^T\boldsymbol{A}$. If it is irreducible, then it has a unique (up to a scalar multiple) dominant Perron eigenvector and the uniqueness problem cannot arise. This follows directly from the Perron–Frobenius Theorem. It ensures that if $\boldsymbol{A}^T\boldsymbol{A}$ is irreducible it possesses an unique dominant positive eigenvector, and that's the one the power method will converge to (can easily be seen in Equation (3.4)).

However, in the case of reducibility, obviously $\boldsymbol{A}^T\boldsymbol{A}$ is not positive by itself nor will there be a positive power of it that is strictly positive. So there is no unique dominant eigenvector and convergence depends on the initial condition.

Be aware that irreducibility of $\boldsymbol{A}$ is *not* sufficient for irreducibility of $\boldsymbol{A}^T\boldsymbol{A}$: A simple example can demonstrate this potential problem. Choosing a cycle of length n (which obviously *is* strongly connected, hence irreducible) corresponds to $\boldsymbol{A} = \boldsymbol{\Pi}_n$, where $\boldsymbol{\Pi}_n$ is a permutation of the identity matrix of order $n \geq 2$. It follows $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{\Pi}_n^T\boldsymbol{\Pi}_n = \boldsymbol{\Pi}_n^{-1}\boldsymbol{\Pi}_n = \mathbf{id}_n$ in this case (as $\boldsymbol{\Pi}_n$ is orthogonal). The identity matrix being reducible, we have an irreducible adjacency matrix $\boldsymbol{A} = \boldsymbol{\Pi}_n$ resulting in a reducible authority matrix $\boldsymbol{A}^T\boldsymbol{A} = \mathbf{id}_n$.

PageRank encounters the same uniqueness problem, and a similar trick as we shall describe for PageRank (cf. Subsection 3.4.2 on page 48) may be applied here order to force uniqueness. Another possibility would be not to randomly initialize the score vectors, but to always start off with $\boldsymbol{x}^{(0)} = \boldsymbol{y}^{(0)} = \mathbf{1}_n/n$, which guarantees convergence to the same limit. A more elegant way however is suggested in [45] using the so–called "exponentiated input to HITS".

Rate of convergence

The convergence and uniqueness issues above as well as the rate of convergence follow directly from general results on the power method, see for example [24]. The iteration equation Equation (3.3) on page 42 in the case where the dominant eigenvalue has multiplicity 1 can be written as follows:

Assume that $\boldsymbol{M} := \boldsymbol{A}^T\boldsymbol{A}$ is irreducible. As $\boldsymbol{M}$ is symmetric, it has n mutually orthogonal eigenvectors $\boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_n$ (corresponding to the n eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ with $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_n|$) which form a basis of $\mathbb{R}^n$. We thus may write the starting vector $\boldsymbol{x}^{(0)}$ as a linear combination

$$\boldsymbol{x}^{(0)} = c_1\boldsymbol{\nu}_1 + \ldots + c_n\boldsymbol{\nu}_n$$

If $c_1 \neq 0$ — in other words demanding that the initial vector is not orthogonal to the dominant eigenvector $\boldsymbol{\nu}_1$ — it follows from Equation (3.3) on page 42 that at iteration step t

$$\boldsymbol{x}^{(t)} = \boldsymbol{M}^t\boldsymbol{x}^{(0)} = c_1\lambda_1^t\left[\boldsymbol{\nu}_1 + \sum_{j=2}^{n}\frac{c_j}{c_1}\left(\frac{\lambda_j}{\lambda_1}\right)^t\boldsymbol{\nu}_j\right] \tag{3.4}$$

Here it is clear that the error is of the order of $(\lambda_2/\lambda_1)^t$ and hence the rate of convergence is dictated by the ratio of λ_2/λ_1 which is particularly important for the PageRank algorithm.

If $\boldsymbol{M}$ had been reducible, Equation (3.4) would have more than one term before the summation spanning the multidimensional eigenspace into which the power method would converge to, depending on the initial condition (starting vector).

3.3.3 Implementation

To close this section on HITS we now take a look at Listing 3.1 which is a simple way of implementing the basic algorithm that calculates the scores for a given network.

The algorithm starts by defining an initial error (needed to enter the `while`–loop in Line 12) and sets in Line 4 the precision to three decimal places more than the minimum the number of nodes would require: if, for example, we have 1000 nodes, we want at least 4 significant figures to be able to properly distinguish between the scores (as all the scores are packed tightly into the $[0, 1]$ interval) — by adding three more figures we are certain to be "on the safe side". We also set $\boldsymbol{x}_{(0)} = \mathbf{1}_n/n$.

Next, $\boldsymbol{A}^T\boldsymbol{A}$ is created (we need to convert $\boldsymbol{A}$ to the `double` format as $\boldsymbol{A}^T\boldsymbol{A}$ needs to hold real numbers).

Then we perform the actual power method (Lines 12–17) with normalisation at each step until sufficient precision is obtained. Finally, we use the simple algebraic connection Equation (3.1) on page 42 to calculate $\boldsymbol{y}$, Line 21.

```
1  function [ x , y ] = hits( A )
2
3  err = 42;               % initial error
4  eps = 10^(-floor(log10(N)+1)-3); % error tolerance
5  x(:,1) = ones(n,1)/n;   % initial auth. scores
6
7  A = double(A);          % make sure A isn't a logical
8  ATA = A'*A;             % create authority matrix
9
10 % power method iterations
11 k = 1;
12 while err > eps
13     k = k + 1;          % increase counter
14     x(:,k) = ATA * x(:,k-1); % update x
15     x(:,k) = x(:,k) / sum( x(:,k) ); % normalize x
16     err = abs(  sum( x(:,k)-x(:,k-1) )  ); % error
17 end
18
19 % calculate y algebraically
20 x = x(:,k);
21 y = A*x; y = y / sum(y);
```

Listing 3.1: *An implementation of the HITS algorithm*

With the implementation of the HITS algorithm we would like to move on to another eigenvector based ranking scheme, namely the famous PageRank algorithm.

3.4 PageRank

Since early 1996 Larry Page and Sergey Brin, at the time two Ph. D. candidates at Stanford University, have been working on a new concept for a search engine for the world wide web, [13, 52], which was first called "BackRub", but later renamed to "Google"[5]. In 1998 they founded Google Inc. as all the other major search engine companies were not interested in their new system. The basis of Google was and remains the PageRank algorithm, which, just like HITS, analyzes the hyperlink structure of web to give a ranking for search results. Let's take a look at the basic concepts and some mathematical background.

3.4.1 Concept

After crawling and indexing the web, PageRank analyzes the link structure to determine the "global" importance of a page. At query time, all query related documents are retrieved from an inverted index and then ranked according to their PageRanks, allowing for an almost instantaneous reply to user queries.

The "raw" google matrix

Basic idea The PageRank measure of importance of a page can be seen as weighted votes from all the other pages, weighted by the importance of the linking (voting) pages themselves and the number of outlinks of those pages:

$$r(P) = \sum_{Q \in \mathcal{B}_P} \frac{r(Q)}{|Q|} \tag{3.5}$$

where $\mathcal{B}_P$ is the set of all pages pointing to page P, $r(Q)$ being the PageRank of page Q and $|Q|$ representing the number of outlinks from page Q. So in order to have a high PageRank, a page P wants to have as many inlinks as possible, quality sites pointing at it with at best no other links on these pages. This makes sense, when we consider that if the inlink is only one out of many it might not be as important as being the sole reference on the linking page.

Iterative approach Because of the recursive definition of PageRank, computation of it necessarily needs iteration. So arbitrarily assign each of the n Pages $P_1, \ldots, P_n$ an initial ranking, say $r_{(0)}(P_i) = 1/n$ and then successively refine their PageRanks by computing

$$r_{(t)}(P_i) = \sum_{Q \in \mathcal{B}_{P_i}} \frac{r_{(t-1)}(Q)}{|Q|}, \qquad \text{for } t = 1, 2, 3, \ldots \tag{3.6}$$

Combining all the PageRanks in a row vector $\boldsymbol{\pi}_{(t)}^T = \begin{pmatrix} r_{(t)}(P_1) & \ldots & r_{(t)}(P_n) \end{pmatrix}$ and creating a normalized hyperlink matrix $\boldsymbol{P}$ with

$$p_{ij} = \begin{cases} 1/|P_i|, & \text{if } P_i \text{ links to } P_j \\ 0, & \text{otherwise} \end{cases}$$

[5] The word "google" is a common spelling of "googol", the term coined by Milton Sirotta. In 1938 he was nine years old and a nephew of the American mathematician Edward Kasner, who asked the boy to find a word for the number 10^{100} ...

again $|P_i|$ being the number of outlinks from page P_i, we may rewrite Equation (3.6) on the preceding page as

$$\boldsymbol{\pi}_{(t)}^T = \boldsymbol{\pi}_{(t-1)}^T \boldsymbol{P}, \qquad \text{for } t = 1, 2, 3, \ldots \tag{3.7}$$

Note that no self–links (self loops) are allowed, resulting in zero entries along the main diagonal of $\boldsymbol{P}$. One recognizes the power method, but this time hopefully converging to a left hand eigenvector of $\boldsymbol{P}$. If the limit exists, the limiting vector $\boldsymbol{\pi}^T$ is then defined as the PageRank vector.

Before moving on to some further interpretation, we will introduce a little example to illustrate the above considerations. Figure 3.2 below shows a small six node web graph.

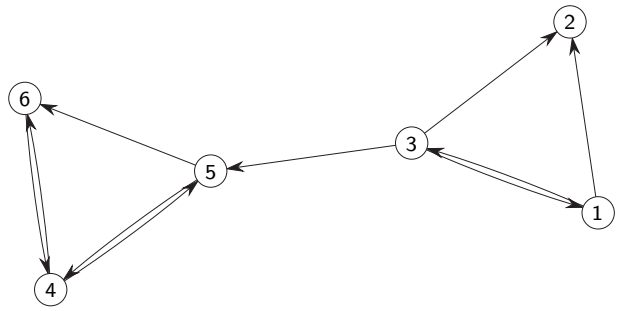


Figure 3.2: *Small web graph for PageRank illustration.*

The normalized hyperlink or "raw" google matrix in this case would be

$$\boldsymbol{P} = \begin{bmatrix} \cdot & 1/2 & 1/2 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1/3 & 1/3 & \cdot & \cdot & 1/3 & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1/2 & 1/2 \\ \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot \end{bmatrix} \tag{3.8}$$

For theoretical and practical reasons, $\boldsymbol{P}$ needs to be adjusted to some extent, which we will discuss below.

Markov Chain

If we look at the raw google matrix $\boldsymbol{P}$ in Equation (3.8) it is nonnegative with row sums equal to one or zero. The latter occurs if there are nodes with no outlinks (also called "dangling"– or "leaf"–nodes), such as node 2 in our example. Assuming, for the moment, that there are no dangling nodes, $\boldsymbol{P}$ is a row stochastic matrix describing the evolution of a finite Markov chain[6].

[6] A common definition for a finite Markov chain, taken from [42], would be a process determined by a finite set of n states $\{S_1, \ldots, S_n\}$ and a set of transition probabilities p_{ij}, $i, j = 1, 2, \ldots, n$. The process can only be in one state at a any time instant. If at time t that process is in state S_i, then at time $t + 1$ it will be in state S_j with probability p_{ij}, an initial starting state being specified.

The Markov chain whose evolution $\boldsymbol{P}$ represents (given that there are no rows with all zeros) can be interpreted as a random walk on the web: a user infinitely surfing the world wide web by randomly clicking on one of the links on each page with equal probability depending on the number of outlinks of the current page. The probability of the surfer on page i clicking on the link leading to page j would correspond to the entry p_{ij}.

As for nonnegative matrices, the dominant eigenvalue is bounded by the minimum and maximum row sums [42], the Perron–eigenvalue of a stochastic matrix is always 1. Consequently, if the PageRank iteration Equation (3.7) on the previous page converges, it converges to the left–hand eigenvector $\boldsymbol{\pi}^T$ of $\boldsymbol{P}$ satisfying

$$\boldsymbol{\pi}^T = \boldsymbol{\pi}^T \boldsymbol{P} \qquad \text{with} \qquad \boldsymbol{\pi}^T \mathbf{1}_n = 1 \tag{3.9}$$

This corresponds to the limiting, stationary or steady–state distribution of the Markov chain [43]. Intuitively, one could think of it as the long term distribution of the time spent on the different pages of our eternally surfing world wide web user.

3.4.2 Computation

As mentioned before, calculating the PageRank comes down to solving the left–hand eigenvector problem Equation (3.9). So if the dangling nodes are taken care of, it seems that all one needs to do is iterate the power method long enough until a satisfying degree of precision is reached. The problem is that, in contrast to HITS, the occurring matrix is not based on a small subset of nodes but on *all* the nodes of the world wide web. Google reported in November 2004 to have indexed more than 8 billion pages, hence the google matrix would be 8–billion–dimensional, [25]. This is the reason why the PageRank calculation has been called "the worlds largest matrix computation" [48].

This is certainly a challenging topic, but as we won't deal with matrices of this size, we do not need to look at the practical issues in handling such a huge matrix and calculations based on it.

Adjusting $\boldsymbol{P}$

At this point we should rather direct our attention to how $\boldsymbol{P}$ needs to be manipulated in order to assure convergence of the power method toward a unique solution.

Taking care of dangling nodes In the above discussion we assumed that $\boldsymbol{P}$ has no rows with all zeros. An easy remedy for this would be to replace the zeros in these rows by $1/n$–entries, where n is, of course, the order of $\boldsymbol{P}$. This modified version $\bar{\boldsymbol{P}}$,is called the transition matrix. An interpretation of this modification would be that our passionate surfer randomly jumps to another page (for example by manually entering its URL in the location bar), once he arrived at a page where he has no links to click on.[7]

[7] Google however uses a more realistic distribution than the uniform one as it is apparently *not* always equally likely for him to jump to *any* other page on the web.

For our small web from Figure 3.2 on page 47 this modification would lead to the following transition matrix

$$\bar{\boldsymbol{P}} = \begin{bmatrix} \cdot & 1/2 & 1/2 & \cdot & \cdot & \cdot \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/3 & 1/3 & \cdot & \cdot & 1/3 & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1/2 & 1/2 \\ \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot \end{bmatrix} \tag{3.10}$$

Fixing reducibility Another problem, that will almost always occur when dealing with real web page data, is that $\bar{\boldsymbol{P}}$ will be reducible. The interpretation of that would be that the Markov chain eventually gets trapped at some stage (in some subgraph), which in return usually will depend on the initial conditions chosen. This is the same uniqueness problem we had when dealing with HITS, cf. Subsection 3.3.2 on page 43.

By reordering the states of the chain, the transition matrix $\bar{\boldsymbol{P}}$ can be made to have the canonical form Equation (A.1) on page 97. There, once a state in $\mathcal{S}_2$ is reached, the chain gets trapped in that set. Usually there are several connected components instead of one single big one (the case of connectedness) and the chain can end up in any one of them, depending on the initial condition chosen.

To fix reducibility and convergence issues, the following convex combination was suggested by Brin and Page as *passe–partout* solution, as it creates a strictly positive matrix:

$$\bar{\bar{\boldsymbol{P}}} = \alpha\bar{\boldsymbol{P}} + (1-\alpha)\frac{\mathbf{1}_{n\times n}}{n} \succ \mathbf{0} \tag{3.11}$$

with $\mathbf{1}_{n\times n}$ being the $n \times n$ matrix with all ones and $\alpha \in [0, 1]$ the so–called "fudge" factor (or more technically the "damping"–factor, as it limits the extent to which a document's rank can be inherited by the documents it links to).

A possible interpretation for α would be the surfer's tendency to sometimes "jump" directly to a different page by manually entering its URL into the address bar. So $(1 - \alpha)$ can be seen as the probability that our passionate world wide web user would "teleport" himself to *any* another page (with equal probability), ignoring possible links on the current page.

This is a special case of the more general formulation

$$\bar{\bar{\boldsymbol{P}}} = \alpha\bar{\boldsymbol{P}} + (1-\alpha)\mathbf{1}_n\boldsymbol{v}^T \tag{3.12}$$

where $\boldsymbol{v}^T$ is the so–called personalization vector. This vector must be a probability vector and each element can be interpreted as the probability that our surfer would "teleport" himself to the corresponding page. A non–uniform distribution makes much more sense, as it is *not* equally as likely to enter any possible URL.

3.4.3 Iterating

Rate of convergence

Using the matrix given by Equation (3.11) has a very nice side effect. As pointed out in Subsection 3.3.2 on page 44, the rate of convergence is gov-

erned by the degree of separation between the dominant and the subdominant eigenvalues. By its nature, the link structure of the world wide web brings $|\lambda_2|$ extremely close to one, [38]. The above convex combination affects the eigendistribution of $\bar{\bar{\boldsymbol{P}}}$ in such a way that if $\mathcal{S}(\bar{\boldsymbol{P}}) = \{1, \lambda_2, \ldots, \lambda_n\}$ is the spectrum of $\bar{\boldsymbol{P}}$, then $\mathcal{S}(\bar{\bar{\boldsymbol{P}}}) = \{1, \alpha\lambda_2, \ldots, \alpha\lambda_n\}$, cf. [37].[8] Thus the asymptotic rate of convergence boils down to how fast α^t tends to 0.

Now this leads us to a well discussed "dilemma". Decreasing α speeds up convergence, but alters the actual transition matrix which results in moving away from the true nature of the web. Increasing α on the other hand dramatically slows down convergence and besides that, sensitivity issues start to appear, as described in Subsection 4.5.2 on page 78. Brin and Page report to use an α around 0.85, [13], which commonly agreed to be a good choice.

Precision

Due to the stochastic nature of $\boldsymbol{\pi}^T$, all the documents' PageRanks need to be tightly packed into a section of the $[0, 1]$ interval. For $n > 8 \cdot 10^9$ an accuracy of *at least* 10^{-9} is needed, resulting in a minimum of about 127 iterations with $\alpha = 0.85$. However, as comparisons at query time will only be made between a small subset of elements of $\boldsymbol{\pi}^T$, less accuracy than that is required. Brin and Page report success using only between 50 to 100 iterations.

In our example from Figure 3.2 on page 47, with $\alpha = 0.9$ one would need at least 23 iterations to reach a precision[9] of 10^{-5}, resulting in

$$\boldsymbol{\pi}^T = \begin{pmatrix} 0.03721 & 0.05396 & 0.04151 & 0.37508 & 0.20600 & 0.28624 \end{pmatrix}$$

and thus a ranking with decreasing PageRank importance of 4, 6, 5, 2, 3, 1.

3.4.4 Implementation

To close this section on PageRank we would like to present a simple implementation of the basic algorithm, which is shown in Listing 3.2 on the facing page.

It can easily be seen that if we use the $\bar{\bar{\boldsymbol{P}}}$ from Equation (3.11) in the power method iteration we can rewrite Equation (3.7) on page 47 as

$$\begin{aligned} \boldsymbol{\pi}_{(t)}^T &= \boldsymbol{\pi}_{(t-1)}^T \bar{\bar{\boldsymbol{P}}} \\ &= \alpha \boldsymbol{\pi}_{(t-1)}^T \bar{\boldsymbol{P}} + (1-\alpha) \boldsymbol{\pi}_{(t-1)}^T \frac{\mathbf{1}_{n \times n}}{n} \\ &= \alpha \boldsymbol{\pi}_{(t-1)}^T \bar{\boldsymbol{P}} + (1-\alpha) \frac{\mathbf{1}_n}{n} \end{aligned} \tag{3.13}$$

which is the actual iteration equation at the heart of our implementation. This way we can split off some work that, effectively, only needs to be carried out once instead of at every iteration step.

[8] In fact, Haveliwala *et al.* prove in [30] that under given circumstances the second largest eigenvalue of $\bar{\bar{\boldsymbol{P}}}$ actually *equals* α.

[9] Precision τ here being the maximum change allowed for any component of $\boldsymbol{\pi}_{(t)}^T$ between iteration steps t and $t-1$, i. e. $\tau = \max_i |\pi_{(t)_i} - \pi_{(t-1)_i}|$

As for the implementation of HITS, we start by defining an initial error, set the precision to three decimal places more than is minimally necessary and initiate $\boldsymbol{\pi}_{(0)}^T = \mathbf{1}_n^T/n$, Lines 3–7.

Next, $\boldsymbol{P}$ is created by converting $\boldsymbol{A}$ to the `double` format (as $\boldsymbol{P}$ also needs to hold real numbers) and transforming it according to Subsection 3.4.2 on page 48: we calculate the sum of each row, replace zero entries in the resulting vector by ones (to prevent divisions by zero) and then use a diagonal matrix formed from it to divide each row of $\boldsymbol{P}$ by its row sum (or just 1 in case of a zero row), Line 17. The dangling nodes are then taken care of by replacing the corresponding rows by $1/n$–entries, giving us $\bar{\boldsymbol{P}}$ in Line 18.

Finally we perform the power method in Lines 22–26 until sufficient precision is obtained, keeping as any operations outside of the `while`–loop as possible.

```
1  function [ p ] = fastpr( A , alpha )
2
3  err = 42;                      % set initial error to some value
4  eps = 10^(-floor(log10(N)+1)-3); % set error tolerance
5
6  one_over_ns = ones(1,N) / N;   % n element vector with 1/n-entries
7  p = one_over_ns;               % initial PR distribution
8
9  % 1. manipulate matrix: adj -> raw google matrix
10 rowsums = sum(A,2);            % calc all the row-sums
11 zrows = find(rowsums==0);      % get indices of all the zero-rows
12
13 % replaces zeros by ones to prevent division by zero in next step
14 rowsums(zrows) = ones(length(zrows),1);
15
16 % devide each row of A by its rowsum (or by 1)
17 P = spdiags(1./rowsums,0,N,N)*double(A);
18 P(zrows,:) = ones(length(zrows),N)/N; % fix dangling nodes
19
20 % 2. perform the power method
21 P = alpha*P; z = (1-alpha)*one_over_ns; i = 1;
22 while err > eps
23    i = i+1; p_old = p;         % increase counter, store old p
24    p = p_old*P + z;            % update p
25    err = max( abs(p-p_old) ); % error
26 end
```

Listing 3.2: *An implementation of the PageRank algorithm.*

A completely different approach to quantifying importance is taken by the following more topologically motivated measures.

3.5 Centrality measures

In the following section, we take a different approach to ranking, an approach that is not an eigenvector based iterative method like the two we have just finished discussing.

We will introduce three different notions of centrality used to identify (at least geometrically) important nodes: excentricity, status and centroid value. These classical measures have been around for many years and extensively studied as they are especially useful in resource allocation problems and transportation networks.

Using notions of centrality to rank nodes in a network is immediate: the more central a node, the more important it must be. Central nodes are the crossroads of a network and may give an intuitive idea of the flows of information. Moreover, in the spirit of graph evolution models, these nodes usually reflect some historical importance, as the network must have grown "around" them.

These considerations suggest the use of centrality measures for ranking, and their usage in biological networks has been recently proposed in [63]. Each of the following three measures will first be motivated by some intuitive example of their use, then defined properly and finally implemented.

3.5.1 Excentricity

Motivation and Definition

Suppose you are an urban planner and it is your task to place a fire station. Of course, you want to place it in such a way that the worst case response time is minimal. In a simple traffic network setting, where nodes correspond to places and edges to the roads connecting them (under the assumption the locations are separated by unit distances), this typical facility allocation problem searches the nodes with *minimal* maximum distance to all others.

The maximum distance from one node to all other nodes is called excentricity:

Definition 3.1 (Excentricity)

*In a connected graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$, the **excentricity** of a vertex $v \in \mathcal{V}$, denoted $e(v)$, is defined as the maximum over all the distances to the other nodes, i. e. $e(v) = \max_{u \in \mathcal{V}} l_{vu}$.*

*The **radius** of the graph, denoted $\rho(\mathscr{G})$, is defined as the minimal excentricity of its nodes, i. e. $\rho(\mathscr{G}) = \min_{v \in \mathcal{V}} e(v)$.*

*The **center** of the graph, denoted $\mathcal{C}(\mathscr{G})$, is the set of vertices with excentricity equal to the graph's radius, i. e. $\mathcal{C}(\mathscr{G}) = \{v \in \mathcal{V} | e(v) = \rho(\mathscr{G})\}$.*

Note that the *maximal* excentricity over all vertices in a (connected) graph is the graph's diameter, $d(\mathscr{G}) = \max_{v \in \mathcal{V}} e(v)$.

We will now see one use of our somewhat different definition of the distance matrix.[10] As we don't always have the "luxury" of a connected graph,

[10] That is that the distance between mutually not reachable nodes is set to n (the total number of nodes), see Subsection 1.3.3 on page 7

our specialized distance matrix allows for a meaningful extension of the above definition to disconnected graphs. It is clear that the smaller the excentricity of a node, the more central it is. However, a node that is "isolated" in some way (that is at least one other node *cannot* be reached from it), will be penalized by the large value n and fall "out of competition" for centrality.

In our example of the "fire station placement problem", its the center set of the traffic network graph that contains all the nodes that solve the problem. It is interesting to note that this set is always contained in a single subgraph, which is a star (that is a connected graph without articulation points[11]), [29].

Calculation

The calculation of the excentricities of nodes follows directly from its definition and is straightforward once we have got the distance matrix `D` of the network. The vector `e` containing the excentricity of each node would then be `e = max(D,[],2)`.

3.5.2 Status

Motivation and Definition

Imagine you are to decide on where to place a new shopping mall. It will be best for business if most people could reach it as easily as possible, i. e. you demand minimum average driving time to it.

A classical resource allocation concept to help you out in this case is called status:

Definition 3.2 (Status)

*In a connected graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$, the **status** of a vertex $v \in \mathcal{V}$, denoted $s(v)$, is defined as the sum over the distances to all the other nodes in the graph, i. e. $s(v) = \sum_{u \in \mathcal{V}} l_{uv}$*

*The **status of the graph**, denoted $s(\mathscr{G})$, is defined as the minimal status of its nodes, i. e. $\sigma(\mathscr{G}) = \min_{v \in \mathcal{V}} s(v)$.*

*The **median** of the graph, denoted $\mathcal{M}(\mathscr{G})$, is the set of vertices with status equal to the graph's status, i. e. $\mathcal{M}(\mathscr{G}) = \{v \in \mathcal{V} | s(v) = \sigma(\mathscr{G})\}$.*

The definition of status was introduced by Frank Harary in a sociometrical context, [28], however the concepts of center and median of a graph were already considered by Camille Jordan in 1869, [33]. In the above example, the "shopping mall placement problem" would be solved by all the nodes in the median set.

Note that instead of the status, one may prefer to consider the average distance between one node and all the other nodes. In case of connected graphs, this is equivalent to the status, as the average distance would simply be the status divided by $e - 1$.

As for excentricity, our modified version of the distance matrix allows for a sensible extension of the above definition to disconnected graphs.

[11] An articulation point (also called cut vertex) of a graph $\mathscr{G}$ is a vertex whose removal disconnects $\mathscr{G}$. Node 1 in Figure 3.3(a) on page 56 is such an articulation point.

Calculation

Again, calculation is trivial once we have the distance matrix `D` of the graph. The vector containing all the stati of the nodes would be `s = sum(D,2)`.

3.5.3 Centroid value

Motivation and Definition

Keeping ourselves busy with placing shops, we are, in a different situation, aware of a competitor who also wants to place a shop. Under the assumption that customers always buy at the nearest shop, where should we place our shop, knowing that the competitor will place his shop at some later stage?

The following considerations give us an answer to that question. Let's model the problem with a graph where the vertices are the customers and the edges correspond to the road network, each edge representing a unit distance. The customers always buy at the closest store and in case of equal distances shopping is equally likely at either store.

For vertices u and v let $\mathcal{V}_{uv} = \{w \in \mathcal{V} | l_{wu} < l_{wv}\}$ be the set of all the other vertices that are closer to u than to v. Letting $\tilde{z}(u,v) = |\mathcal{V}_{uv}| - |\mathcal{V}_{vu}|$, it is clear the we should place our shop in at the location u that maximizes $\tilde{z}(u,v)$ over all possible locations v for our competitor, who will then naturally choose a node v that minimizes $\tilde{z}(u,v)$, so $s(v) \longrightarrow \min_{v \neq u} s(v)$.

Entringer *et al.* prove in [18] that $s(u) + |\mathcal{V}_{uv}| = s(v) + |\mathcal{V}_{vu}|$ for all connected graphs. With this identity we can write $\tilde{z}(u,v) = -s(u) + s(v)$, an the solution to the shop placement problem would be to look for vertices u for which $\tilde{z}(u,v) = -s(u) + \min_{v \neq u} s(v)$ is maximal.

This motivates the definition of the centroid value:

Definition 3.3 (Centroid value)

*In a connected graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$, the **centroid value** of a vertex $v \in \mathcal{V}$, denoted $z(v)$, is defined as $z(v) = s(v) - min_{u \neq v} s(u)$.*

*The **centroid value of the graph**, denoted $\chi(\mathscr{G})$, is defined as the minimal centroid value of its nodes, i. e. $\zeta(\mathscr{G}) = \min_{v \in \mathcal{V}} z(v)$.*

*The **centroid** of the graph, denoted $\mathcal{C}(\mathscr{G})$, is the set of vertices with centroid value equal to the graph's centroid value, i. e. $\mathcal{C}(\mathscr{G}) = \{v \in \mathcal{V} | c(v) = \zeta(\mathscr{G})\}$.* —

Note that this definition is different from the one Peter J. Slater gave in [57], namely in that the sign of $\tilde{z}(u,v)$ has been inverted. The reason for that is purely aesthetic and was suggested in [63]: in analogy to excentricity and status we would like this third centrality measure to be minimal at the most central nodes. In fact, contrary to the other measures, the centroid value of a node might very well become zero, or even negative.

If in a connected graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$ we find a node $\bar{v}$ with centroid value $z(\bar{v}) = 0$, it implies that $s(\bar{v}) = \min_{u \neq \bar{v}} s(u) = \sigma(\mathscr{G})$. Hence, by definition, $\bar{v} \in \mathcal{M}(\mathscr{G})$ and there exists at least one other node with the same minimal status, i. e. $|\mathcal{M}(\mathscr{G})| \geq 2$.

However we can find $z(\hat{v}) < 0$ only for *at most* one node $\hat{v}$. In this case, $s(\hat{v})$ is the *unique* minimum of all the stati, and $\mathcal{M}(\mathscr{G}) = \mathcal{C}(\mathscr{G}) = \{\hat{v}\}$.

Calculation

Due to the fact that for each node we need to determine the minimum of the status over all *other* nodes we cannot calculate the vector containing all the centroid values with a single command; we have to use a `for`–loop. First we calculate all the stati and store them in `s`. Then, in the loop, we remove at each step the current node from `s` and finally search for the minimum, see Listing 3.3 below.

```
function z = centroid( D )

z = zeros(n,1);       % initialize
s = sum(D,2);         % calculate all the stati
for i = 1:n
   tempstat = s;      % temporarily store s
   tempstat(i) = []; % remove current
   z(i) = s(i) - min(tempstat);
end
```

Listing 3.3: *Function returning the centroid values of all the nodes in a graph.*

Yet another, radically different approach to measuring importance is taken by our last ranking scheme, namely the one induced by "damage".

3.6 Damage

We would like to close this chapter by introducing a seventh measure: damage. This measure has been introduced in [39] for bipartite graphs, and more recently for undirected graphs in [56].

3.6.1 Motivation and Definition

In biological networks for example, another idea of the importance of a node (enzyme) would be the effect its removal would have. One missing enzyme could prevent the production of a number of proteins and thus have a critical effect on the entire organism.

The damage value of a node has been introduced to quantify the effect of the deletion of that particular node. Similar to [56] we define the damage value of a node as follows:

Definition 3.4 (Damage)

In a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, let $\mathcal{G}_v = (\mathcal{V}_v, \mathcal{E}_v)$ be the connected component that contains node v, so $v \in \mathcal{V}_v$, and let $\tilde{\mathcal{G}}_v = (\tilde{\mathcal{V}}_v, \tilde{\mathcal{E}}_v)$ be the largest connected component of $\mathcal{G}_v$ after the removal of node v, so $v \notin \tilde{\mathcal{V}}_v$.

Then the value

$$\partial(v) = |\mathcal{V}_v| - |\tilde{\mathcal{V}}_v|$$

is the ***damage value*** *(or just* ***damage****) of node v.*

So the greater the damage, the greater the difference in sizes of the cluster node v was in before its removal and the largest "piece" that is left of it.

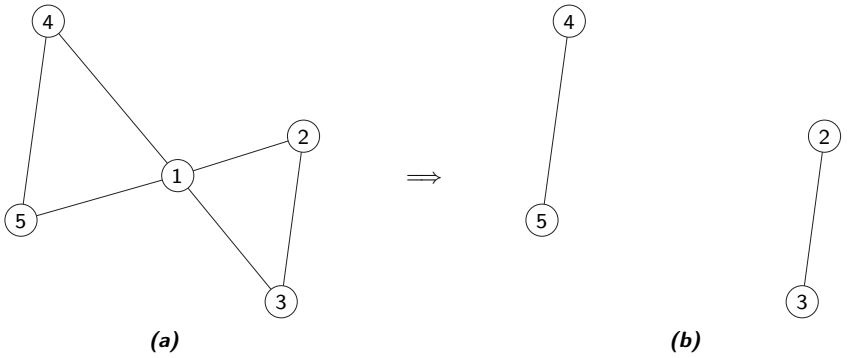


Figure 3.3: *Illustration of the amount of damage the removal of node 1 can cause.*

If in contrast a node v has damage $\eth(v) = 0$, it means that the node was isolated in the first place, so its removal has no effect at all. If $\eth(v) = 1$, the damage caused by its removal is limited to the disappearance of the node itself, leaving the rest of the network intact (either the node was a "dead end" by itself, or, if it connected two other nodes, an alternate route is present).

To illustrate the definition of damage, take a look at Figure 3.3. In (a), the size of the connected component that contains node 1 is 5. After the removal of the cut vertex 1, the remaining largest piece has size 2 as we can see in (b), so $\eth(1) = 5 - 2 = 3$. However, the damage of node 5 for instance would be only $\eth(5) = 1$, as its removal does not disconnect the graph (or, more precisely, the connected component it is in).

3.6.2 Calculation

The calculation of damage of a node is unfortunately an extremely costly task, as for each node we have to determine twice the largest connected component of a (sub)graph in order to determine its size.

The function `damage` works hand in hand with the function `splitintocc`, which we shall describe first.

Function `splitintocc`

As its name suggests, the function splits a graph into its connected components. More precisely, the function returns two cell arrays which contain respectively an adjacency matrix for and a vector containing the names of the nodes in each connected component.

The adjacency matrices however are a bit special, in that they are not obtained by "chopping off" (removing) all the nodes that are not part of a particular connected component (and hence changing the size of the matrix, and the "names" of the nodes), but just by setting the corresponding rows and columns to zero. That way, the adjacency matrices keep their original sizes,

```
function [ ccsA , ccsN ] = splitintocc( A )

% calculate matrix Asum2. nonzero entries correspond to mutually
% reachable nodes (see function  avl  for more details).
n = length(A); nn = n^2; m = 0; Δnnz = 42;
Asum2 = logical(speye(n)); Am = Asum2;
warning('off','MATLAB:conversionToLogical'); % we know that issue.

while (nnz(Asum2) ≠ nn) && (Δnnz ≠ 0) && (m ≤ n)
   m = m + 1; Am = logical(double(Am)*A); % new power of A
   Asum1 = Asum2; Asum2 = Asum1 | Am;     % add it to Asum2
   Δnnz = nnz(Asum2)-nnz(Asum1);          % increase of new paths
end

ncc = 0; ccsA = {}; ccsN = {}; oneton = 1:n; rmved = [];

% now go through the connected components
while ¬isempty(Asum2)
   ncc = ncc + 1;

   % grab all the nodes that are reachable from first node ...
   thiscc = find(Asum2(1,:));

   % take care of node name changes resulting from the removals ...
   origind = oneton; origind(rmved) = []; % "simluate" removal
   thiscc_origind = origind(thiscc);
   removeme_origind=oneton; removeme_origind(thiscc_origind)=[];

   % now create the different adjacency matrices of the different ccs
   ccsA{ncc} = A;                         % take A ...
   ccsA{ncc}(removeme_origind,:) = false; % set non-cc-members to 0
   ccsA{ncc}(:,removeme_origind) = false; % ...
   ccsN{ncc} = thiscc_origind;            % and store their names

   % finally get them out of the way
   Asum2(thiscc,:) = []; Asum2(:,thiscc) = [];
   rmved = [rmved,thiscc_origind];
end
```

Listing 3.4: *Function returning the connected components (and the respective nodes in these) of a graph.*

the nodes keep their "names" and only the nodes that are part of the current connected component keep edges attached to them.[12]

Much of the functionality is similar to our `avl` function, see Subsection 1.4.2 on page 12. Basically, we create a matrix `Asum2` which contains a nonzero element (i, j) if node i can be reached from node j, which is then used to determine the connected components.

However, special care needs to be taken in the process: as (in order to save work) we remove nodes along the way, we must keep track of the original names of the nodes.

[12] This has a practical reason, because now `slcc(ccsA{x})` will automatically report the correct size of the connected component of the nodes in `cssN{x}`

So we first "simulate" the removal of the nodes (Lines 25–27) and that way obtain the original names the "surviving" nodes had before the others got removed.

With these precautions we can detect a connected component, Line 22, restore the original "names" of its members, Line 27, and finally create the special adjacency matrix for it in its appropriate cell of `ccsA`, Lines 30–33. This done, the component gets removed from the network and the process restarts.

Function damage

Now that we have a means of getting the connected components for each node, calculating the damage values of the nodes is quite straightforward, following immediately from its definition.

The algorithms basically has two modes. Either you want to know the damage values of some (specified) nodes — in this case you have to pass the function a *connected* graph or the type of adjacency matrix produced by `splitintocc` — or you want to know the damage values of all the nodes in the graph (in which case `splitintocc` is run automatically).

Note t5hat `slcc` is an alias function that runs `avl` to determine the size of the largest connected component.

```
1  function [ d ] = damage( A , nodes )
2
3  if nargin == 1 % no particular nodes specified
4     nodes = [];
5  end
6
7  d = [];
8
9  if isempty(nodes) % we must do all the work.
10    [G,nodes] = splitintocc(A);
11    for j = 1:length(G)
12       d(nodes{j}) = damage(G{j},nodes{j});
13    end
14 else
15    n = length(find(sum(A,2)));   % slcc(A)
16
17    for i = 1:length(nodes)
18       A_temp = A; % make temporary copy of A
19       A_temp(nodes(i),:) = [];   % delete row
20       A_temp(:,nodes(i)) = [];   % delete col
21       d(i) = n - slcc(A_temp);   % calc damage
22    end
23 end
```

Listing 3.5: *Function returning the damage of some or all nodes in a graph. Note that the function* `splitintocc` *is shown in Listing 3.4 on the previous page.*

With this last measure of importance we would like to close this chapter on ranking schemes and shall continue by investigating their robustness with respect to changes in the graphs they are applied to.

CHAPTER 4

Robustness

We devise 4 types of network perturbations and 6 different ways of measuring deviations in rankings, then present quantitative results as well as some theoretical results on the ranking's robustness.

4.1 Introduction

Now that we have discussed different measures suitable for ranking nodes in a network, we should investigate the robustness of the resulting rankings in particular with respect to variations in the network.

The data gathered in real life is often based on some physical measurements. This implies that the data might be inaccurate, incomplete or incorrect to some degree, reasons for that possibly being insufficient precision in the physical measurements, the influence of noise, or deliberate sampling (given that measuring 100% of the system may require unjustifiabilly more time or money).

In any case, it is important to know how sensitive the rankings are to various types of perturbation. A ranking scheme that inverts the order of the nodes when only a few edges are changed will be of little practical use. To analyse the impact of perturbations, we introduce a number of ways to measure the deviations between two rankings. We then perturb the network by incrementally removing, adding or rewiring edges, or removing nodes — as these are phenomena close to real life problems — and compare the deviation between the rankings of the original and the perturbed networks.

For the simulations we used randomly generated Barabási–Albert graphs as their degree distribution resembles those of protein–protein interaction networks that we are going to look at in the last chapter. Our extensive quantitative results are also accompanied by a few theoretical results concerning PageRank and HITS.

4.2 Perturbations

As mentioned above, having data which is 100% correct is a rather "utopian" case. Being able to identify important nodes by some ranking is one thing, having a ranking that is also robust to missing or slightly wrong data is another.

Note that perturbing edges has in two ways a rather limited effect, compared to changing nodes. On the one hand, changing one edge only affects two nodes

(but may disconnect the graph) and there are usually much more edges than nodes. On the other hand, removing a node also implies the removal of all edges joining it, so the impact will usually be much greater.

We shall now describe various types of possible perturbations.

4.2.1 Edge perturbations

Edge removal

One of the most intuitive perturbations is surely edge removal. By that, we mean the random deletion of a number of edges.

This would correspond to the case for example, where measurements missed out on some interactions between proteins.

Edge addition

The dual case to edge removal would be edge addition in random positions, similar to the procedure used to generate Erdös–Rényi graphs.

Such a perturbation could be introduced for example by measurement noise or wrong interpretation of certain results.

Edge rewiring

In the case of edge rewiring, the number of edges stays constant (this time similar to the generating algorithm of the Watts–Strogatz model).

This type of perturbation can also be induced by noise or misinterpretations of lab results.

4.2.2 Node perturbations

Node removal

Another intuitive perturbation would be (random) node removal, i. e. we randomly choose a number of nodes and remove them (as well as all the edges joining it).

This would correspond to missing out on some (maybe intermediate) reactant or additional "ingredient" in some reaction due to either inaccurate measurements or deliberate sampling of data.

Node attack

By node attack we mean the systematic and targeted removal of the most important nodes. This is a rather artificial alteration, as it implies some intentional act (and the *a priori* knowledge of the importance of all the nodes). We therefore only mention it here, but will not examine it further in this present work.

Node addition

Node addition would be the random addition of nodes, which must also be accompanied by the addition of a number of edges connecting those new nodes to the network. As in the biological networks we shall consider later the number of nodes (proteins) is usually well known (through, for example, sequencing the genome), it is very unlikely that some perturbation would actually add nodes to the network. This is the main reason why we shall also leave this perturbation out of the following investigations.

4.3 Measures of deviation

We shall now present six different approaches to measuring the deviations between the ranking of an original and a perturbed network. We will call these "Method 1 – 6". At the beginning of each we try to give a short and condensed question to motivating it.

All methods have in common that they look at the top 5% (plus ties) of the nodes in the rankings. We do this mainly for two reasons:

(i) In practice, one is only interested in a few top ranked nodes, not the entire ranking (just like a person searching some information on the internet only checks the first few results a search engine has come up with, so it is most important to know the sensitivity of these results).

(ii) In case of node removal, the total number of nodes and hence "contributions" to the deviation measure decreases. For a meaningful comparison, however, we need a *constant* number. So, for instance, if we only use a small (constant) number of top ranked nodes (say a number corresponding to 5% of the origininal network size), we can then remove up to 95% of the nodes and still be comparing the same number of deviations.

The first three measures focus on the actual ranks of the nodes. The resulting deviation measure is calculated by taking the L_1–norm of the respective deviations in ranking of the nodes, i. e. we sum up the absolute values of the differences in places. Conversely, the last three measures are based purely on the fact whether a node is in the top 5% of the rankings or not.

4.3.1 Using the ranking of the nodes

The following three methods use the *ranks*[1] of nodes to measure the deviation.

Method 1

> *"How strong an impact has the perturbation on the identification of highly ranked nodes in the 'real' network?"*

To answer that question, we compare the ranks of a number of *initially* top ranked nodes (i. e. in the ranking of the unperturbed network) with where they "ended up" *after* the perturbation.

[1] Please note the way we set up our ranking, which is described at the beginning of Subsection 4.3.4 on page 65.

Table 4.1(a) on the current page illustrates this process. Here we have a network with 10 nodes and look at the top 50% of the nodes. The measure of deviation δ_1 would be calculated as follows:

$$\delta_1 = \underbrace{|1-6|}_{\text{Node 3}} + \underbrace{|1-2|}_{\text{Node 8}} + \underbrace{|3-9|}_{\text{Node 4}} + \underbrace{|4-4|}_{\text{Node 10}} + \underbrace{|4-1|}_{\text{Node 1}} = 15$$

Original		Perturbed	
Rank	Node	Rank	Node
1.	3	1.	1
1.	8	2.	8
3.	4	3.	·
4.	10	4.	10
4.	1	5.	·
6.	·	6.	3
7.	·	7.	·
8.	·	8.	·
9.	·	9.	4
10.	·	10.	·

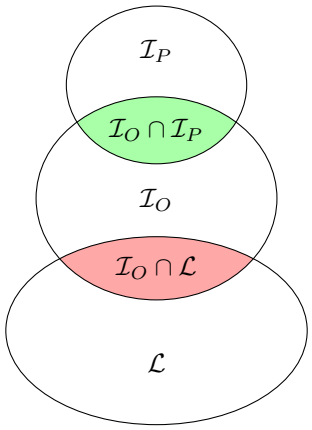
(a) Method 1.

Original		Perturbed	
Rank	Node	Rank	Node
1.	7	1.	9
2.	·	2.	7
2.	4	3.	1
4.	1	4.	4
5.	9	5.	2
6.	·	6.	·
7.	·	7.	·
7.	·	8.	·
7.	·	9.	·
10.	2	10.	·

(b) Method 2.

Original		Perturbed	
Rank	Node	Rank	Node
1.	6	1.	5
2.	2	2.	·
3.	10	3.	·
4.	·	4.	3
5.	·	5.	·
5.	·	6.	6
7.	·	7.	·
8.	·	8.	2
9.	3	9.	10
10.	5	10.	·

(c) Method 3.



(d) Methods 4, 5 and 6.

Table 4.1: *Illustrations for the five methods used to calculate deviations. The dots are placeholders for values that are not used / unimportant.*

Method 2

"How important really are nodes of high rank in the perturbed network?"

This question can be answered by looking at the positions of a number of top ranked nodes from the perturbed graph and compare these with their original ("real") ranks.

In Table 4.1(b) on the facing page we illustrate these comparisons for a fraction of 50% of the top ranked nodes in the perturbed network. The resulting method of deviation δ_2 is then calculated with:

$$\delta_2 = \underbrace{|1-5|}_{\text{Node 9}} + \underbrace{|2-1|}_{\text{Node 7}} + \underbrace{|3-4|}_{\text{Node 1}} + \underbrace{|4-2|}_{\text{Node 4}} + \underbrace{|5-10|}_{\text{Node 2}} = 13$$

Method 3

"How big are the maximum deviations between the rankings?"

To quantify this, we first determine all the deviations and then sum up a fraction of the biggest deviations.

Looking at a fraction of 50% of the deviations, the dashed gray arrows in Table 4.1(c) on the preceding page correspond to those that are not taken into account as they are smaller then the deviations marked by the solid arrows. We determine δ_3 in this case with:

$$\delta_3 = \underbrace{|1-6|}_{\text{Node 6}} + \underbrace{|2-8|}_{\text{Node 2}} + \underbrace{|3-9|}_{\text{Node 10}} + \underbrace{|9-4|}_{\text{Node 3}} + \underbrace{|10-1|}_{\text{Node 5}} = 31$$

4.3.2 Using the number of correctly identified nodes

Method 4

"What's the chance that a seemingly important node is, in fact, not important?"

To answer this question, we first define two sets: $\mathcal{I}_O$ is made up of the top 5% *plus ties* of the (original) ranking and $\mathcal{I}_P$ of the top 5% *plus ties* of the ranking of the perturbed network.[2]

Then we look at the cardinalities of the sets and their intersection. The nodes that are identified as important and that *really are* important would be found in the set $\mathcal{I}_O \cap \mathcal{I}_P$. If we compare the number of nodes in this set with the total number of "important" nodes in the perturbed network ($\mathcal{I}_P$), we get an idea of the quality of the ranking.

To set up a deviation measure, now let

$$\delta_4 = 1 - \frac{|\mathcal{I}_O \cap \mathcal{I}_P|}{|\mathcal{I}_P|}$$

[2] See Table 4.1(d) on the facing page for an illustration of the sets. The numbers 5% and 5% here are chosen arbitrarily to some extent, and they do not necessarily have to be identical.

This measure can then be interpreted as the probability that a seemingly important node in the perturbed network is, in fact, *not* really important.

Of course, this measure strongly depends on the chosen "limits" for importance. In the following, we will stick to the suggested top 5% from both the original and the perturbed ranking.

To give a little example, say we take the ranking from Table 4.1(a) on page 62. Considering 40% and 20% of the top ranked nodes as "important", we would have $\mathcal{I}_O = \{3, 8, 4, 10, 1\}$ (note that even though 40% correspond to 4 nodes, we also took node 1 in as place 4. is a tie) and $\mathcal{I}_P = \{1, 8\}$. Hence $\mathcal{I}_O \cap \mathcal{I}_P = \{1, 8\}$ and finally $\delta_4 = 1 - 2/2 = 0$ indicating a flawless performance of the "ranking scheme" behind the two presented rankings.

Method 5

> *"What's the probability that an important node does not get identified as such?"*

In other words, we now would like to have an idea of how many of the nodes that really are important do not stay in a certain top fraction of the ranking after perturbation.

An attempt to answer this question is quite similar to our previous measure. We define the sets $\mathcal{I}_O$ and $\mathcal{I}_P$ in the same way as above (i. e. for example top 5% plus ties) and then let

$$\delta_5 = 1 - \frac{|\mathcal{I}_O \cap \mathcal{I}_P|}{|\mathcal{I}_O|}$$

So the only difference is that we divide by the number of elements in $\mathcal{I}_O$ instead of in $\mathcal{I}_P$.

4.3.3 Method 6

> *"What's the chance that the importance of top ranked nodes gets totally misjudged?"*

An example for such a misjudgment would be if originally important nodes end up at the bottom of the ranking of the perturbed graph. So an approach to the question above would be to define an other set, $\mathcal{I}_L$, which contains the, say, bottom 25% of the nodes in the ranking of the perturbed graph. We can then introduce

$$\delta_6 = \frac{|\mathcal{I}_O \cap \mathcal{I}_L|}{|\mathcal{I}_O|}$$

That way, δ_6 corresponds to the probability that an originally important node ends up in the bottom quartile of the network.

4.3.4 Possible problems

Ties in the ranking

With the exception of PageRank and HITS, it is very likely in the other ranking schemes that a (sometimes considerable) number of nodes gets exactly the same "score". For example, one just has to look at the node degrees, where (in the scale–free networks we are going to use) a multitude of nodes will have degree 3.

Because of this we will use a ranking that respects ties: The rank of a node corresponds to 1 + the number of nodes that perform strictly better than the node itself. Such a ranking (base on node degrees in this example here) could look like this:

Rank	1.	1.	3.	4.	5.	5.	7.	...
Node	4	43	1	38	51	14	7	...
Degree	5	5	4	3	2	2	1	...

When nodes disappear

Some precautions need to be taken in case of node removing perturbations: When we remove nodes, the adjacency matrix not only changes in size, but the nodes do not correspond to their initial row– and column indices anymore.

To overcome this, we need to keep track of which nodes have been removed and then remove the corresponding entries from the vector the ranking algorithm produces.

In practice, this is done the following way. When we want to compare for example the PageRank values between two matrices — where the first matrix corresponds to the original network, and the second one to the perturbed network (with dimensions `n2 = n1-length(rmved)` due to the removal of nodes `rmved`) — we first calculate the PageRanks for both matrices. Before comparing both of them however, we remove the values corresponding to nodes `rmved` from the (longer) PageRank vector `p1` of the first matrix: `p1(rmved) = []`. That done, we can easily calculate the deviations by some suitable sorting of `p1` and `p2`.

We shall now move on to the actual quantitative evaluation of the robustness of the rankings schemes presented in the previous chapter, using the deviation measures we have just finished describing.

4.4 Quantitative assessment

Having reviewed the deviation measures and some potential problems, we would now like to evaluate experimentally the robustness of the ranking algorithms.

4.4.1 Preliminary remarks

Before presenting our results we should mention a few important points.

Our setup

Due to the number of combinations that result from the different deviation measures (6), different perturbations (4), different ranking schemes (7), different graph models (3) and different network parameter combinations (countless), we cannot present all of our results.

The first step in reducing this overwhelming number of possible combinations is that we limit ourselves to only looking at a one type of graph and the (fixed) sets of generating parameters: In all of the following, we used scale–free graphs on $n = 3000$ nodes and roughly $e \simeq 9000$ edges — generated with the Barabási–Albert model using $n_0 = m = 3$.

We chose this model not only for reasons of simplicity and the fact that it is well studied, but mostly because many networks encountered in biology show scale–free characteristics (see Subsection 2.5.2). The actual parameter choice is of course arbitrary to some extent, but produces graphs similar in size to the real world graphs we will be looking at in the next chapter.

For each deviation method we will then show the impact of the different perturbations by comparing the sensitivity of each of the seven ranking schemes. Any of the following values and plots represent the average from 250 independent runs of the respective simulation.

The results are plotted using the following markers:

×	Node degrees	△	Excentricity
○	HITS	◊	Status
□	PageRank	•	Centroid value

For the PageRank rankings we used the standard $\alpha = 0.85$.

Disconnection

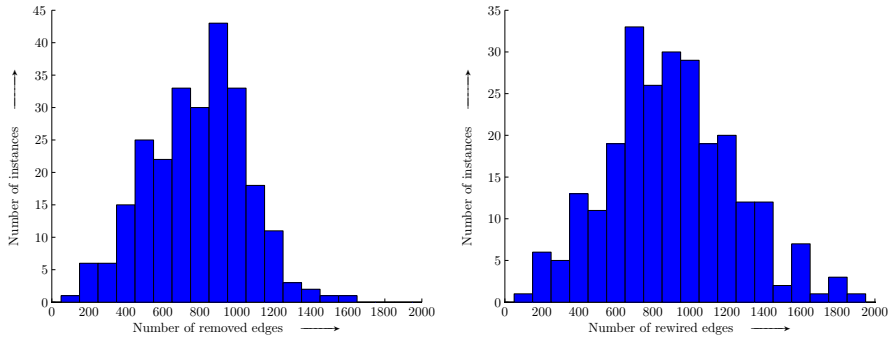
The scale–free graphs we are generating are always connected (this is guaranteed by the Barabási–Albert model we will use). However, when we start to remove or rewire edges, sooner or later the graph will get disconnected.[3]

As the disconnection of the graph will significantly influence the centrality based ranking algorithms it is important to study the point at which the graphs usually become disconnected. We did so by generating 250 scale–free graphs and perturbed them in a similar increasing way (in the exact same way as used for the comparisons below). We then took note after how much perturbation the graph became disconnected.

Figure 4.1 on the next page shows the distributions from these simulations. There, we can see in (a) that the amount of perturbation required for disconnecting the graphs can be as little as 100 or as high as 1600 removed edges. From the 250 graphs generated, the calculated mean was around 785 removed edges (with standard deviation of ≈ 266) needed to disconnect the graph.

A similar result holds for edge rewiring, see (b). Disconnection can occur in a broad interval ranging from 100 to 1900 rewired edges which contains the calculated mean value of ≈ 900 (with standard deviation ≈ 345).

[3] This is immediate to see for edge removals; in the rewiring case we could argue that the graph is turning more and more into an Erdös–Rényi graph. For these graphs, 9000 edges for 3000 nodes are far from enough to "guarantee" connectedness, see Theorem 2.1 on page 22.



(a) Edge removal *(b) Edge rewiring*

Figure 4.1: *Distribution of disconnected graphs for increasing perturbations. A total of 250 graphs were independently generated to average.*

We have not included a plot for node removal, as removing up to 100 nodes never disconnected a single graph (in over 500 trials).

Status and centroid value

In all plots and tables we will see that status and centroid value produces seemingly exactly the same results. This indicates that the rankings induced by both rankings must be very close, if not identical.

This does not come as a surprise, as it can be easily explained by looking at their definitions again: they only differ by the "$-min_{u \neq v} s(u)$" term which is constant (but for at most one single node). Thus, both rankings will be quite if not totally the same.

Excentricity and damage

We shall see below that excentricity often behaves strangely with respect to perturbations. This is surely due to the fact that excentricity gave very clustered ranking on our networks. The scores where usually concentrated on three or four values at the most, so the rankings are basically three or four different places that are shared by a large number of nodes.

A representative example would be for instance where out of the 3000 nodes 15 would have excentricity 4 (putting them in place 1), 1463 nodes with excentricity 5 would be in place 16 and 1521 nodes had excentricity 6 and ended up in place 1479.

Similar, but even worse, is the situation for damage. It is not only significantly more costly to simulate, but also not helpful as a method — at least for the type of graphs we are studying. These are connected by design and their connectedness is also quite robust to single node removals removing (which is done in order to calculate the damage, see Section 3.6 on page 55).

To illustrate these two points we plotted the average distribution of scores in Figure 4.2 on the next page. This graphic shows the average scores of the 3000 nodes returned by the different ranking schemes, normalized and sorted in

increasing order. We can see that for most of the nodes, the excentricity curve is absolutely flat, meaning that many nodes get the same score. The situation is even more dramatic for the damage scores, see Figure B.2 on page 101, where *each and every* node has the same score.

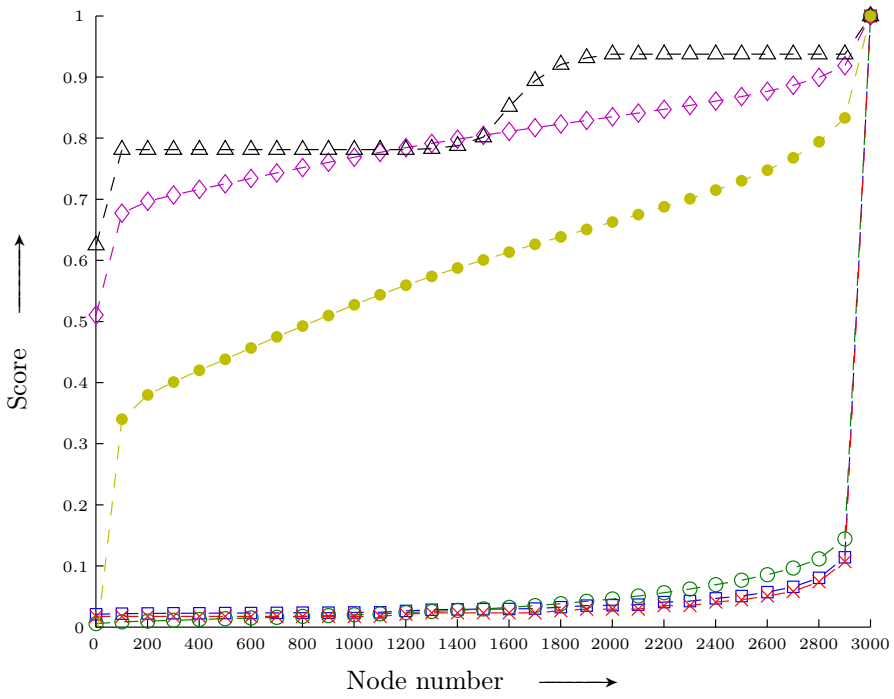


Figure 4.2: *Average distribution of the scores of the different ranking schemes on 200 Barabási–Albert graphs with $n = 3000$ nodes and $n_0 = m = 3$.*

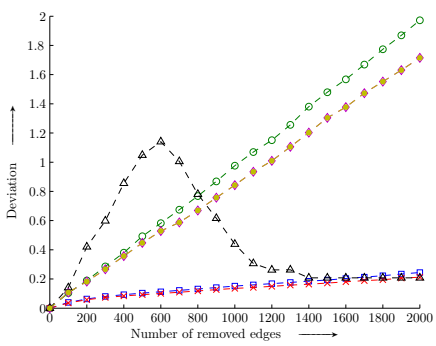
With these considerations in mind we still kept excentricity and damage in our simulations[4], solely for reasons of completeness. We shall present and analyse the results from these simulations.

4.4.2 Results

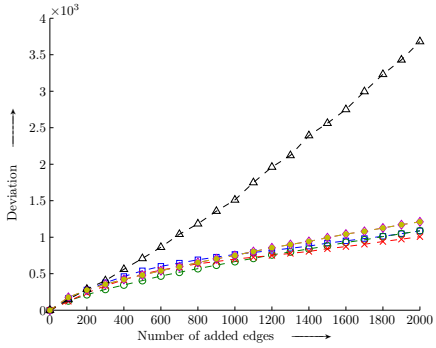
Method 1

General remarks Figure 4.3 on the facing page shows the relevant plots for this first measure. Note that we cannot present results for node removal perturbations as some of the originally most highly ranked nodes might get removed with increasing perturbation. In that case we cannot perform the necessary comparisons.

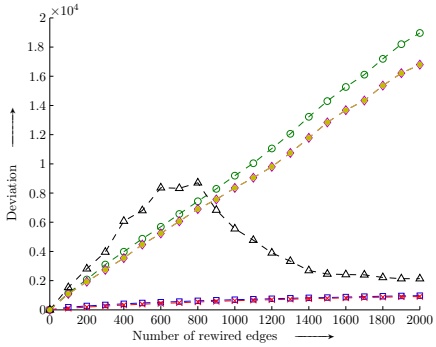
[4] The results for damage, however, have only been calculated for smaller graphs of 150 nodes; these results are shown in the Appendix B on page 99



(a) Edge removal



(b) Edge addition



(c) Edge rewiring

Figure 4.3: *Sensitivity for deviation measure 1.*

Besides some (explainable) nonlinear behavior for excentricity the other curves seem quite linear. But in the case of edge addition, PageRank and node degrees clearly perform best compared to the other measures.

Edge removal One of the most remarkable things in Figure 4.3(a) is probably the shape of the curve for the excentricity (△). The error seems to increase first but then decreases again after about 600 links have been removed. This behavior however does not reflect the real sensitivity of the rankings, as it can be explained in the following way.

In the early stages, as edges get removed, the perturbation increases and initially top ranked nodes might "climb down" the ranking. Obviously, the more edges get removed, the more likely it is that the graph becomes disconnected. As mentioned in Subsection 4.4.1 on page 66 the majority of the graphs usually became disconnected after about 600 removed nodes.

Now, what happens the moment time the graph becomes disconnect? One thing is, that the distance matrix gets its first set of "penalty entries" with the high value $n = 3000$. As a matter of fact, from that moment on, each row and each column in the distance matrix must have at least one penalty entry. As mentioned above, the excentricity of a node corresponds to the maximum entry

in the distance matrix along its corresponding row. Hence suddenly all nodes get the same excentricity, get ranked (shared) first place and the deviation falls down to a low, constant value.

The shape of the curve is then produced by the superimposition of the 100 series of deviation values that are used to average out the results. Each graph in it becomes disconnected at a different point, similar to the shape of Figure 4.1(a) on page 67.

Edge addition We can see in Figure 4.3(b) that the excentricity performs much worse than all the other ranking schemes, which are all roughly equally as good.

However, the absolute deviations are much smaller (by more than one order of magnitude) compared to the other perturbations. This might be due to the fact that the graph cannot become disconnected here.

Edge rewiring When it comes to rewiring, we can see in Figure 4.3(c) that PageRank and node degrees seem to be the best choice as their rankings seem to be much more stable than the other ones. The order of magnitude of the deviations is surprisingly almost precisely the same as for edge removal.

Also, the same shape for the excentricity curve is manifest. The graphs are becoming disconnected, as they gradually turn into Erdös–Rényi random graphs with 3000 nodes and 9000 edges, which will typically not be connected (in average). We thus have the same disconnection issue as for edge removal and a similar explanation for the shape of the curve applies.

Method 2

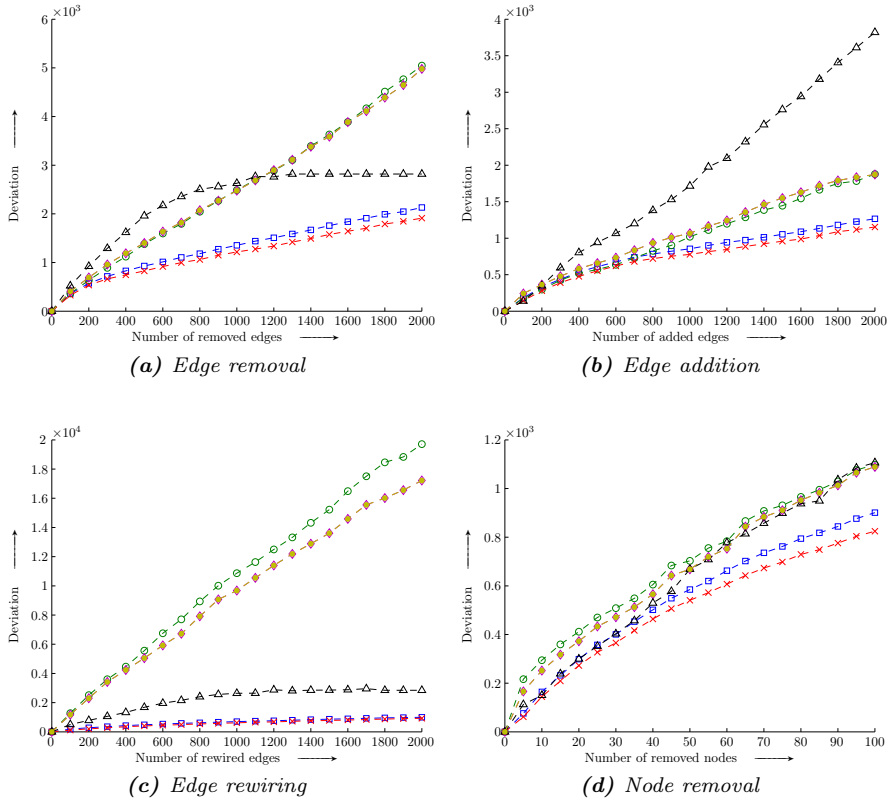
General remarks The relevant plots for this first measure are shown in Figure 4.4 on the next page. There, we can see that most values, again, increase quite linearly, with the exception of excentricity.

Also, PageRank and node degrees do the best job for each type of perturbation, most clearly in case of edge removal and rewiring. If we look closely, the node degree induced ranking seems to be even better than PageRank.

Edge removal For very few disturbances, Figure 4.4(a) shows that all ranking methods seem to perform more or less equally well. From about 200 removed edges on, however, PageRank and node degrees clearly separate from the others. HITS looks equally sensitive as centroid value and status.

Excentricity, again, is showing a bit of an odd behavior. The "bowing off" to the side is surely not an indication that, at some stage, excentricity induced rankings are more robust to the perturbations than HITS, status and centroid value, but rather the result of the disconnection problem which sends all the nodes to (shared) first place and hence results in a constant deviation, as discussed above.

Edge addition At very early stages, all six measures show roughly the same performance, see Figure 4.4(b). Soon, however, differences appear and PageRank and node degrees take the lead.



(a) Edge removal *(b) Edge addition*

(c) Edge rewiring *(d) Node removal*

Figure 4.4: *Sensitivity for deviation measure 2.*

Edge rewiring HITS, being the most sensitive ranking method here, as well as status and centroid value are clearly outperformed by PageRank and node degrees, as we can see in Figure 4.4(c).

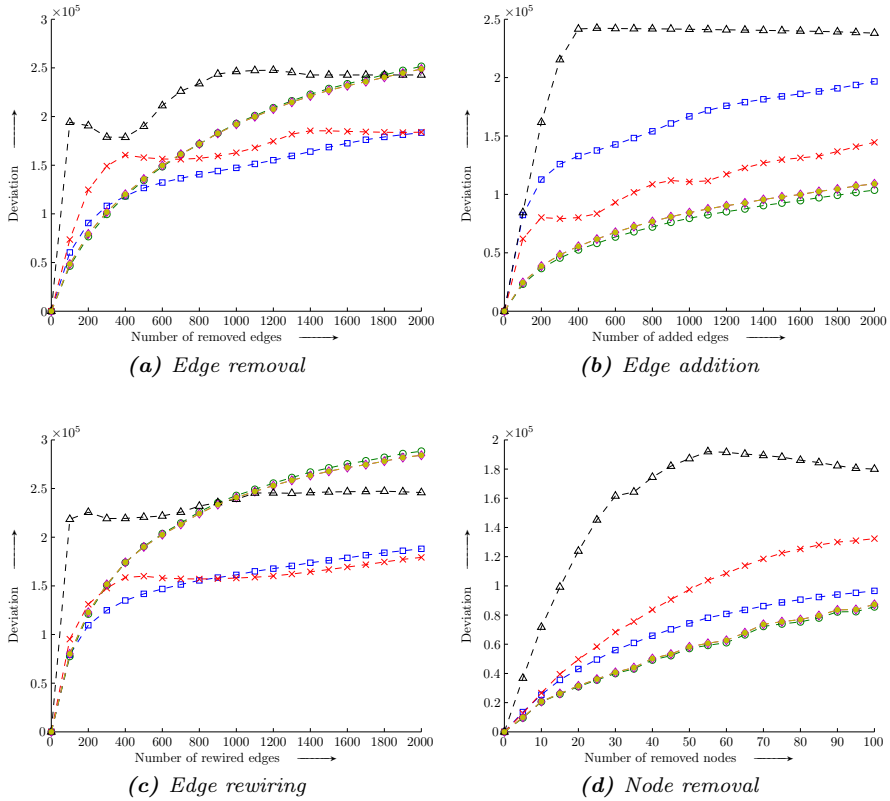
The apparent good performance of excentricity should not be given too much weight in light of its poor resolution, and also that it is effectively useless once the graph becomes disconnected.

Node removal The interesting thing in Figure 4.4(d) is that excentricity does quite a competitive job. In fact, on this plot there is no clear winner (even though PageRank and node degrees perform slightly better than the other measures).

Method 3

General remarks The four plots for this measure are shown in Figure 4.5 on the following page. Out of all the plots, this one shows probably the strangest behaviors.

In plots (a) and (b) the curves for node degree induced ranking show some oscillatory behaviour. To properly explain this further investigations should be carried out.



(a) Edge removal *(b) Edge addition*

(c) Edge rewiring *(d) Node removal*

Figure 4.5: *Sensitivity for deviation measure 3.*

In contrast to the previous two figures, most of the curves on these plots are rather nonlinear. Excentricity also shows some strange behavior in case of edge removal and rewiring. The deviation seems to rise quickly in the first step, decrease slightly and then continue to grow slowly.

HITS appears to be joining the pair of related measures centroid value and status, as their deviations are virtually identical here.

The order of magnitude of the deviations is, as one would expect it, much higher than in the previous cases, as we are picking the maximum deviations to calculate the measure.

Edge removal In early stages of Figure 4.5(a), it is HITS, centroid value and status that perform best. However, from about 400 removed edges on, they are outperformed by PageRank, and a bit later also by the node degree induced ranking.

The sensitivity of node degree induced ranking shows the mentioned oscillatory behaviour.

Edge addition As seen before, HITS, centroid value and status seem to be doing quite a robust job when it comes to adding edges. We can see in

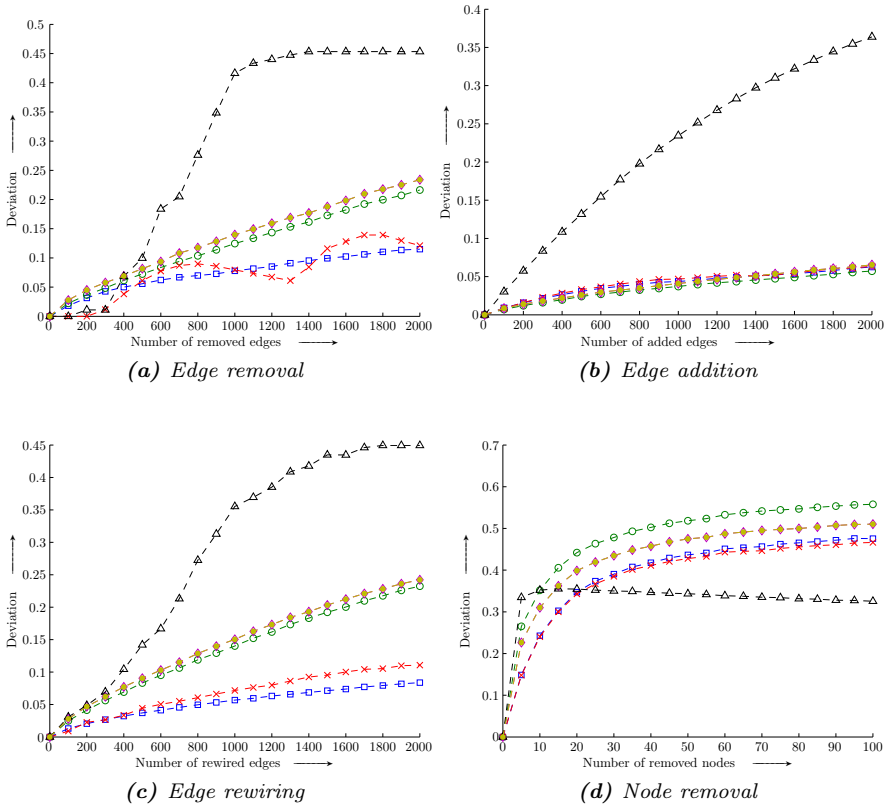
Figure 4.5(b) that independent of the amount of perturbation, they clearly beat PageRank, excentricity and node degrees (which show an oscillatory behaviour).

Edge rewiring In Figure 4.5(c) we get roughly the same picture as for edge removal, however, the drop after the first peaking for excentricity is less pronounced.

PageRank, again, performs well, but the node degree induced ranking seems to overtake it at roughly 1000 rewired edges.

Node removal The plots for node removal, shown in Figure 4.5(d), indicate that here again HITS, centroid value and status preform best, followed by PageRank. Excentricity is out of competition as it seems to be much more sensitive than the other measures.

Method 4



(a) Edge removal *(b) Edge addition*

(c) Edge rewiring *(d) Node removal*

Figure 4.6: *Sensitivity for deviation measure 4.*

General remarks We shall now move on to a radically different measuring scheme. Figure 4.6 shows some of the results from our simulations for deviation measure 4.

The most homogeneous picture is obtained for edge addition, where all the measures except excentricity are more or less equally good.

HITS again sticks quite closely to centroid value and status, PageRank and node degrees perform well with roughly the same amount of deviation for each of the different edge perturbations.

It is interesting to note that PageRank (and in a way also node degrees, if we neglect the oscillations) show quantitatively the same amount of sensitivity for all three types of edge perturbations.

Edge removal Until up to 400 removed edges, excentricity and node degree seem to be better than the other ranking methods, as we can see in Figure 4.6(a). But beyond that point the node degree deviation shows some oscillations again. PageRank then seems to give the most consistent results.

Edge addition There is not much to note in Figure 4.6(b) other than that excentricity preforms poorly again, the other five measures seeming much more robust. They perform roughly equally well, HITS being the best, on close inspection.

Edge rewiring Similar to edge removal, three groups are apparent in Figure 4.6(c). That is, PageRank and node degrees perform best, followed by HITS, centroid value and status, and the rather sensitive excentricity.

Node removal In the case of node removal, Figure 4.6(d), after a fast increase in deviation the curves level out to a more slow increase. The excentricity induced ranking seems to slowly decrease in sensitivity again after the initial "boost", which should not be taken too serious in light of our earlier remarks.

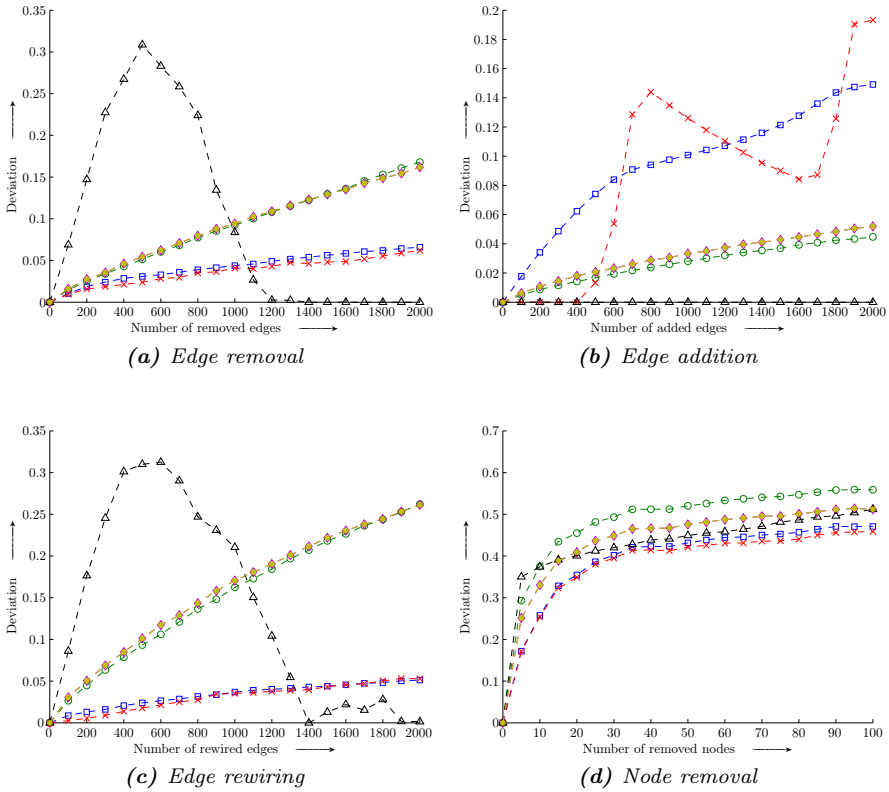
Method 5

General remarks With minor differences, the in shapes of the subplots for edge removal and rewiring in Figure 4.7 on the facing page closely resemble each other as well as to the corresponding ones from measure 1 (Figure 4.3 (a) and (b) on page 69). Here, the shapes of the excentricity curves are very similar to the distribution of disconnected graphs, Figure 4.1 on page 67, which can be explained easily.

Besides the strong oscillations in the sensitivity of the node degree ranking in case of edge addition, both PageRank and node degree perform best in three out of the four cases.

Edge removal The picture is quite clear in Figure 4.7(a), PageRank and node degrees being the methods of choice.

Edge addition Again, we find some oscillatory behaviour in the curve for node degrees in Figure 4.7(b). The ranking induced by the nodes' excentricities seems to be completely robust; next in line would be HITS followed closely by centroid value and status.



(a) Edge removal *(b) Edge addition*

(c) Edge rewiring *(d) Node removal*

Figure 4.7: *Sensitivity for deviation measure 5.*

Edge rewiring The same comments as for edge removal can be made for Figure 4.7(c).

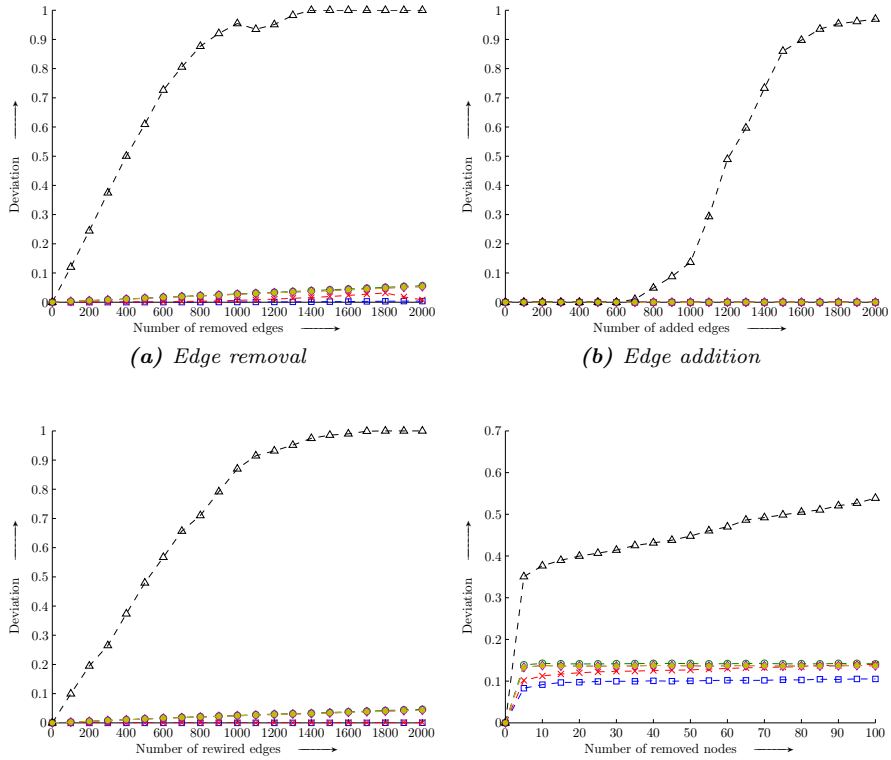
Node removal Clearly, Figure 4.7(d) is very similar to Figure 4.6(d), so we find also for measure 5 that node degrees and PageRank allow for the least sensitive ranking in case of node removal.

Method 6

General remarks It is not hard to see that excentricity does a very bad job in all four cases of Figure 4.8 on the next page. The other measures perform much better, if not ideally well: They appear to be completely robust with respect to edge additions for instance as deviations there are actually zero.

PageRank seems one more time to be the overall winner, followed by node degrees.

Edge removal Besides the strong deviation for excentricity in Figure 4.8(a) (which can be explained by the disconnection issue again), HITS, centroid value



(a) Edge removal *(b) Edge addition*

(c) Edge rewiring *(d) Node removal*

Figure 4.8: *Sensitivity for deviation measure 6.*

and status form the usual group, but PageRank and node degrees perform better than these, PageRank being near perfect.

Edge addition Perhaps surprisingly, Figure 4.8(b) indicates perfect robustness of all the measures other than excentricity.

Edge rewiring Node degrees and PageRank are flawless here as well, as their deviations are constantly zero in Figure 4.8(c). The HITS, centroid value and status trio however shows a slightly increasing deviation.

Node removal Interestingly enough, all measures show sort of a "jump" to a certain deviation value where they then stay more or less constant, Figure 4.8(d). PageRank is again the seemingly most robust measure.

4.4.3 Conclusions

Once again, we would like to point out the additional plots in Appendix B on page 99. Generally speaking, we can say that in most of the cases PageRank and node degrees seem to be the most robust ranking methods. There are

however cases, where HITS as well as centroid value and status challenge this position, this happening most frequently at edge adding and node removing perturbations.

With the sensitivity varying from measure to measure it is hard to give a general recommendation, even without taking into account the actual usefulness of the produced rankings themselves. In other words: the best scheme strongly depends on the question asked.

Before closing this chapter, let us quickly mention two theoretical results on sensitivity.

4.5 Theoretical results

As a complement to the extensive quantitative assessment of sensitivity we would now like to present two theoretical results, one on HITS and one on PageRank, which have been published in a paper by Ng *et al.*, [51].

4.5.1 Sensitivity of HITS

They claim (and prove) that the stability of the HITS algorithm is governed by the so–called eigengap of the authority matrix, that is the difference between the largest and the second largest eigenvalue of $\boldsymbol{A}^T\boldsymbol{A}$.

This is done through the following theorem, which is unfortunately only limited to very small perturbations, that is addition or removal of a number of edges to/from exactly one node.

Theorem 4.1 (Sensitivity of the HITS algorithm)

Let $\boldsymbol{A}$ be the adjacency matrix of a graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$, and $\boldsymbol{M} = \boldsymbol{A}^T\boldsymbol{A}$ be its associated authority matrix with principal eigenvector $\boldsymbol{\nu}_1$ and eigengap $\eta = \lambda_1 - \lambda_2$. Assume the maximum out–degree of every node of $\mathscr{G}$ is bounded by $k_{\max}$.

For any $\varepsilon > 0$, suppose we perturb $\mathscr{G}$ by adding or deleting at most k links from one node, where

$$k < \left(\sqrt{k_{\max} + \alpha} - \sqrt{k_{\max}}\right)^2 , \qquad \text{where } \alpha = \frac{\varepsilon\eta}{4 + \sqrt{2}\varepsilon}$$

Then

$$\|\boldsymbol{\nu}_1 - \tilde{\boldsymbol{\nu}}_1\|_2 \leq \varepsilon$$

holds for the perturbed principal eigenvector $\tilde{\lambda}_1$ of the perturbed matrix $\tilde{\boldsymbol{M}}$.

Proof: [51]. □

As we know from Subsection 3.3.2 on page 42, the principal eigenvector contains the hub scores of the network, so as little a change as possible in it is desired. However, knowing a bound on the amount of change in this vector

does not imply a bound on the deviation of the ranking the scores induce. It is therefore hard to draw further conclusions from this result.

The same limitation is also valid for their result on PageRank, which we describe now.

4.5.2 Sensitivity of PageRank

Stated in the same paper, the following theorem by Ng *et al.* is significantly less restrictive than the one for HITS:

Theorem 4.2 (Sensitivity of the PageRank algorithm)

Let $\bar{\boldsymbol{P}}$ be the transition matrix of a graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$ and $\boldsymbol{\pi}^T$ be the principal left hand eigenvector of $\bar{\bar{\boldsymbol{P}}} = \alpha\bar{\boldsymbol{P}} + (1-\alpha)\mathbf{1}_{n\times n}/n$. Let nodes $v_1, v_2, \ldots, v_m$ be changed in any way and let $\tilde{\bar{\boldsymbol{P}}}$ be the resulting (new) transition matrix.

Then the new PageRank scores $\tilde{\boldsymbol{\pi}}^T$ satisfy

$$\|\tilde{\boldsymbol{\pi}}^T - \boldsymbol{\pi}^T\|_1 \leq \frac{2\sum_{i=1}^{m} \pi_{v_i}}{1-\alpha}$$

where π_{v_i} denotes the PageRank score of node v_i.

Proof: [51]. □

So, the impact of the perturbation depends on the nodes that are perturbed, and on the "damping factor" α chosen.

It is quite intuitive that the more important the perturbed nodes were in the first place, the bigger the deviation will be resulting from the perturbation. Just think of the internet surfer illustration from above — the more important a page is, the more frequent it will be visited, hence the greater the overall impact of any change to it.

This is another aspect of the dilemma related to α (see Subsection 3.4.3 on page 49). Here as well, one is tempted to lower it as it will certainly decrease sensitivity to perturbations. This becomes most obvious in the radical case of setting $\alpha = 0$. Here, the PageRank algorithm would run purely on the static $\mathbf{1}/n$ matrix, completely ignoring any change in the transition matrix that represents the graph.[5]

4.5.3 Limitations

Simulations not displayed here have shown that the bounds are very conservative. The actual deviation was usually by at least one order of magnitude smaller than the bound.

But most importantly, both results have a major limitation: they only give bounds for a norm of the difference of the score vectors, not the resulting ranking. As this is what we are interested in in our context we have little use here for the theorems.

They are however a good start and point in the right direction, and further investigations should be carried out on this topic.

[5] This also implies that the bound is not tight.

We also noticed that both bounds are very crude. For example the effective deviations resulting from all meaningful perturbations we could inflict on various types of graphs were roughly ten times smaller than the bound given in Theorem 4.2 on the preceding page, indicating that the bound could certainly be improved with further investigation.

Now that we have an idea of the ranking schemes' robustness with respect to differnet perturbations we shall investigate the actual correlation between attributed importance through high ranking and importance in the context of an application, such as protein–protein interaction networks.

CHAPTER 5

Application to real data

After analysing three real world networks we evaluate the ranking schemes' ability on them to identify essential nodes, as well as the robustness of this ability with respect to wrong data.

5.1 Introduction

In the previous chapters we have examined a number of ranking schemes and studied their properties on computer generated networks. However, we have not yet looked at how well these schemes actually identify "important" nodes in real world networks.

Of course, the precise interpretation of "important" depends on the application considered. For that reason, we look in this chapter at the possible application of ranking algorithms to biological networks, namely protein–protein interaction networks.

Even in small and simple organism there are easily thousands of proteins active in a single cell. Proteins can be enzymes directly involved in chemical reactions of the metabolism, or can be part of larger modules like ribosomes or mitochondria. Other types of proteins are involved in regulation and control of gene expression, or serve as messengers that signal between cells.

For many applications it is crucial to understand the interactions between the proteins in living cells. One approach biologists have taken is to establish so called protein–protein interaction networks which allow an effective analysis of the system of interactions. In these networks, proteins form the nodes, and they are linked together if they interact in some way or other, resulting in an undirected graph.

With more and more datasets becoming available we shall evaluate the performance of our ranking schemes in identifying essential nodes.[1] We use three recent datasets, namely those from Uetz *et al.* [59], Yu *et al.* [64] and Ito *et al.* [32].[2] These datasets provide us with the actual networks and also with a list of proteins known to be essential.

After examining the networks we will evaluate the fraction of essential nodes among a certain top fraction of nodes in the different rankings, as well as the robustness of the identification process with respect to perturbations in the networks.

[1] A protein is considered essential if its removal causes the organism to die.

[2] Although the data itself has been taken from the accompanying website of the article by Uetz *et al.* [64]: http://bioinfo.mbb.yale.edu/network/essen/ .

5.2 Examination of the data

The networks we consider here are not connected but rather contain one major connected component and a few, much smaller "satellites". Hence establishing the graphs diameter or calculating its average path length will not give very meaningful values. Also, using measures like excentricity will not provide good rankings.

For these reasons, we will focus our investigations only on the largest connected component within these networks. More precisely, we shall now have a look at the degree distributions and characteristic values, comparing them with Barabási–Albert graphs of comparable size. This is done dataset by dataset.

5.2.1 Uetz dataset

The dataset by Uetz *et al.*, [59], is the smallest of those examined here. It

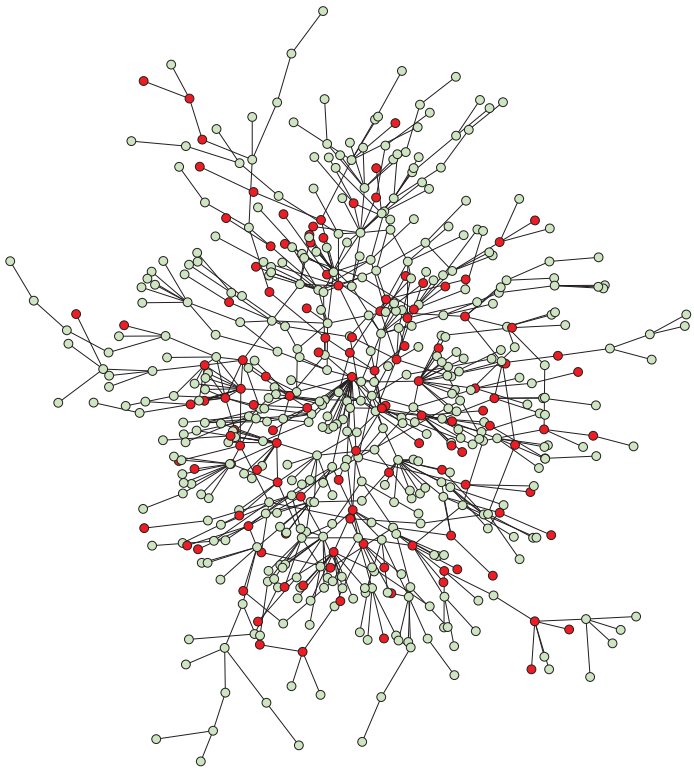


Figure 5.1: *Largest connected component from the Uetz dataset. Essential nodes are marked in red.*

originally contains 1044 proteins and 981 interactions. Its largest connected component however includes only about 54% of those nodes, that is $n = 558$

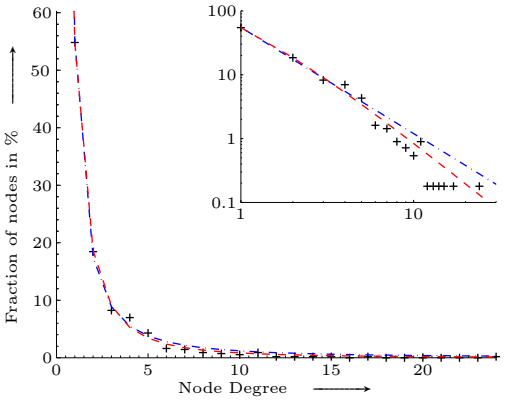
proteins.[3] With $e = 646$ edges it is rather sparsely connected (only 0.42% of the potential reactions take place).

Figure 5.1 on the facing page shows a visual representation of the largest connected component in the Uetz dataset. As we will be interested in essential nodes in Section 5.3, we marked the nodes known to be essential in red.

Degree distribution

As suggested in Subsection 2.5.2 on page 36, protein–protein interaction networks usually have some sort of power law degree distribution. However, we should mention that this is by no means established in a theoretical way. Best results were obtained by fitting a slightly modified $p(k) = a \cdot (k + k_0)^{-\gamma}$ on the actual distribution.[4]

Figure 5.2(a) shows the degree distribution for the largest connected component together with two fitted power laws. The red (dashed) line corresponds to $p(k) \sim (k + 7.1)^{-3.6}$, calculated using the nonlinear fitting function in MATLAB. The fit of the standard power law Equation (2.7) on page 28 returned $\gamma = 1.7$.



(a)

Value	Ito	BA_1	BA_2
$\langle k \rangle$	2.315	1.996	3.989
$\langle l \rangle$	7.285	6.364	3.808
d	18.000	15.720	7.000
$\langle c \rangle$	0.052	0.000	0.042

(b)

Figure 5.2: ***(a)*** *Degree distribution of the largest connected component in the Uetz dataset. The dashed lines show the fitted power laws.*
(b) *Characteristic values of the dataset compared to the average values for 200 comparable Barabási–Albert graphs on $n = 558$, with parameters $n_0 = m = 1$ ("BA_1") and $n_0 = m = 2$ ("BA_2").*

Characteristic values

In Figure 5.2(b) above there is a table of the characteristic values for the largest connected component of the Uetz dataset together with those for two comparable Barabási–Albert random random graphs (that is the average value from 200 instances of each).

[3] The second largest connected component contains only 24 nodes.
[4] The benefits of this modified approach are much more obvious for the Ito and Yu datasets, see for example Figure 5.4(a) on page 85.

The Barabási–Albert model used had $n = 558$ and $n_0 = m = 1$ (column "BA_1"). This gave the fixed power law exponent $\gamma = 3$ and $e = 557$ edges (compared to 646 edges in the dataset). Using $n_0 = m = 2$ (column "BA_2") we obtain graphs with about twice as many edges, $e = 1113$.

It is the lack of edges in the "BA_1" case that explains the smaller average node degree compared to the dataset. The missing clustering is due to the specific generating algorithm (which cannot produce any "triangles" starting with only one node and adding 1 edge with each new node). It is interesting to see, however, that average path length and diameter are smaller — even if there are less edges in total. This means, that the pure Barabási–Albert model creates more efficiently connected graphs.

The much denser "BA_2" setup also fails to properly reflect the properties of the dataset as it is significantly denser. Even so, clustering is lower than in the original data, indicating that the biological network is quite clustered.

5.2.2 Ito dataset

We now take a look at the next larger dataset, namely the one by Ito *et al.*, [32].

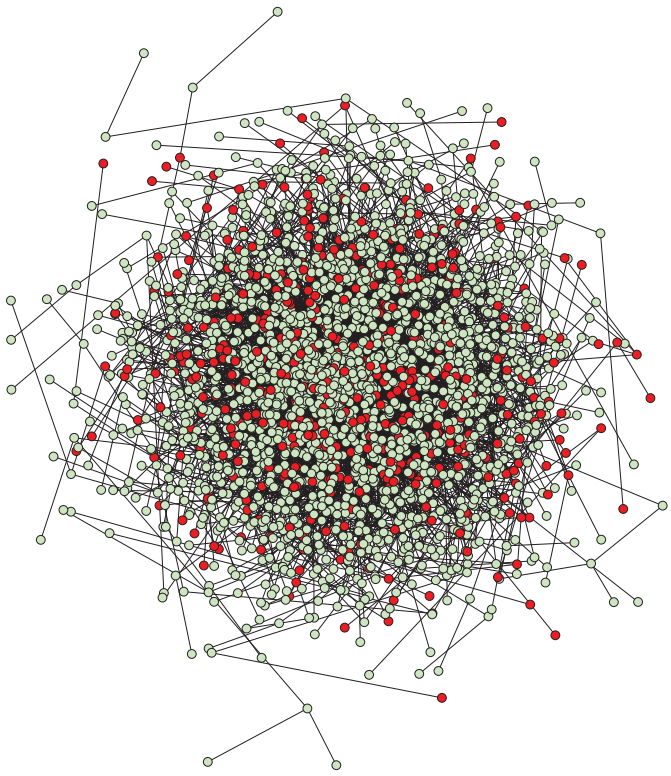


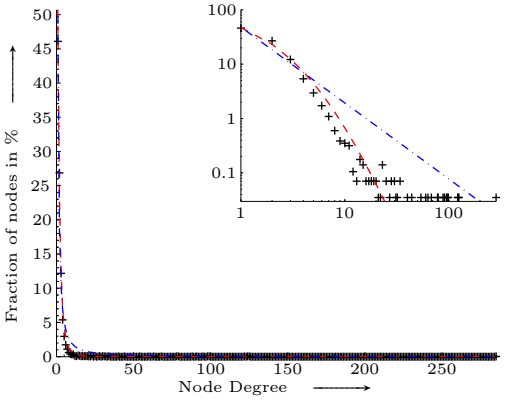
Figure 5.3: *Largest connected component from the Ito dataset. Essential nodes are marked in red.*

It contains a total of 3278 nodes and 4393 edges. Again, the corresponding graph is disconnected and contains one dominating cluster. This largest connected component contains about 87% of the proteins, that is $n = 2840$.[5] A total of $e = 4147$ edges within that component make it the sparsest of the three networks, as only 0.10% of the possible edges are present.

Even so, the visual representation of the network, Figure 5.3 on the preceding page, looks rather dense. Essential nodes are, again, coloured red.

Degree distribution

We show in Figure 5.4(a) below the degree distribution for the largest connected component of the Ito dataset as well as fitted power laws. The best fit was obtained using the modified power law (red dashed line), resulting in $p(k) \sim (k + 5.1)^{-4.7}$. The pure power law resulted in an exponent of $\gamma = 1.4$ (blue dash–dotted line).



Value	Ito	BA_1	BA_2
$\langle k \rangle$	2.920	1.999	3.997
$\langle l \rangle$	4.886	7.972	4.514
d	14.000	20.630	8.010
$\langle c \rangle$	0.038	0.000	0.012

(a) *(b)*

Figure 5.4: ***(a)*** *Degree distribution of the largest connected component in the Ito dataset. The dashed lines show the fitted power laws.*
(b) *Characteristic values of the dataset compared to the average values for 200 comparable Barabási–Albert graphs on $n = 2840$, with parameters $n_0 = m = 1$ ("BA_1") and $n_0 = m = 2$ ("BA_2").*

Characteristic values

The table in Figure 5.4(b) above shows the results from the analysis of the real data as well as two attempts to imitate its properties with a scale–free graph.

As the number of edges of the real graph (more precisely of the largest connected component) falls exactly between the number of edges possible with the standard Barabási–Albert model, we present, again, two different parameter combinations (that is $n_0 = m = 1$ and $n_0 = m = 2$), resulting respectively in graphs with $e = 2839$ (column "BA_1") and $e = 5677$ (columns "BA_2").

We observe similar behavior as for the Uetz dataset: for the sparser graphs ("BA_1"), the average node degree is, of course, lower and clustering is zero. The

[5] All other connected component have equal or less than 6 nodes.

average values of average path length and diameter are however significantly smaller compared to the real network.

The denser graphs "BA_2" have also smaller diameters and average path lengths, but even though it is much denser, clustering is about three times lower than in the real network.

5.2.3 Yu dataset

The largest of our datasets is the one by Yu *et al.*, [64].

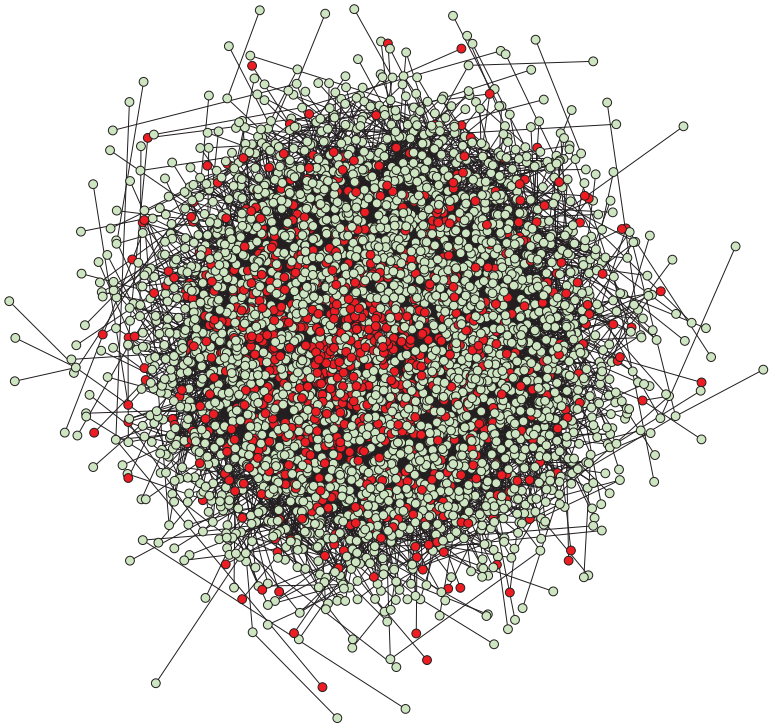


Figure 5.5: *Largest connected component from the Yu dataset. Essential nodes are marked in red.*

It contains records for a total of 4743 proteins and 22980 interactions. Note that this figure includes 629 self–loops. We decided to drop these self–loops, as to better fit the graph in our previously established framework (we would also encounter problems when applying the PageRank algorithm, or when rewiring edges, as we will do in the following two sections).

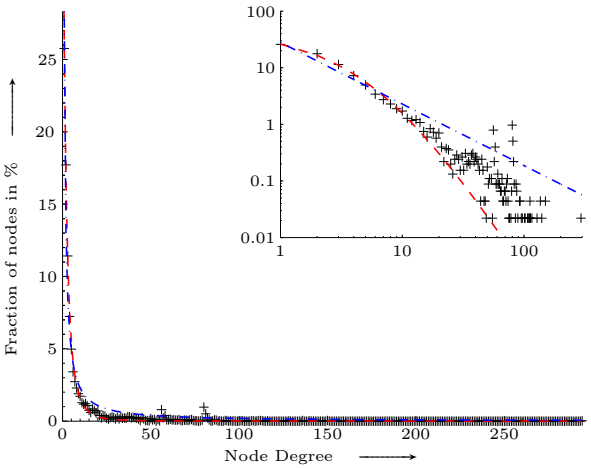
The largest connected component contains about 96% of the nodes, more precisely $n = 4544$.[6] After removing all self–loops, the cluster has $e = 22588$ edges, resulting in a density of about 0.22%.

[6] The second largest connected component contains only 4 nodes.

Figure 5.5 on the facing page depicts the largest connected component of the Yu dataset.

Degree distribution

The degree distribution for this dataset together with the fitted power laws are shown in Figure 5.6(a) below. Again, the modified power law allows for a better fit (red dashed line), and we find $p(k) \sim (k + 5.7)^{-3.3}$. The "blunt" power law fits best with $\gamma = 1.1$ (blue dash–dotted line).



(a)

Value	Ito	BA
$\langle k \rangle$	10.065	9.993
$\langle l \rangle$	4.358	3.429
d	11.000	5.050
$\langle c \rangle$	0.305	0.014

(b)

Figure 5.6: ***(a)*** *Degree distribution of the largest connected component in the Yu dataset. The dashed lines show the fitted power laws.*
(b) *Characteristic values of the dataset compared to the average values for 200 comparable Barabási–Albert graphs on $n = 4544$ ($n_0 = m = 5$).*

Note that the two little "outliers" around $k = 58$ and $k = 81$ may be due to measurement error; this also underlines that the power law is only an model to approximate reality.

Characteristic values

In this situations we find that setting $n_0 = m = 5$ actually produces a graph very similar not only in size but also in density (with a target $n = 4544$ the Barabási–Albert model produces $e = 22705$, which is only 200 edges more than the real network has).

The characteristic values calculated for the largest connected component in the dataset along with those from similar scale–free graphs are shown in Figure 5.6(b) above.

As expected, the average node degrees are close together. We find again, however, that on the one hand the artificial networks seem to be more efficient in connecting nodes (the average path length is smaller, and the diameter is only half of that of the real network) but on the other are significantly less clustered, that is 20 times less.

5.2.4 Conclusions

All three models had in common that their degree distribution deviates quite clearly from a pure power law distribution, but a slight modification can improve the fit.

The networks based on the real data all show a significantly higher clustering than the ones created by the Barabási–Albert model. These, however, feature shorter average path lengths and a smaller diameter.

With the three datasets introduced we shall now continue by investigating the performance of our ranking schemes in identifying biologically essential nodes.

5.3 Identifying essentiality

In the third chapter we presented a number of ranking schemes and motivated their use. Chapter 4 then investigated their robustness with respect to wrong or missing data using artificial networks of the Barabási–Albert type. We now examine the *biological relevance* of the different notions of importance. In particular, we consider question of the type "If a node is PageRank–important, is it also essential for the survival of the organism?"

We will try give an answer to this type of question by examining the overlap or intersection of nodes identified as important by our ranking schemes and nodes known to be essential (from a biological point of view). This is done, again, dataset by dataset.

5.3.1 Our setup

In the following we shall not take into account the actual *rank* of the nodes, as we have only the "binary" information whether a protein is essential or not. In fact, we proceed similar to deviation measure 4 from the previous chapter.

Calculation of quality

Recall Table 4.1(d) on page 62, with the sets $\mathcal{I}_O$ and $\mathcal{I}_P$, and think of them, now, as being respectively the set of "known–to–be–essential"–nodes and the set of "thought–to–be–important"–nodes given by the ranking scheme.[7]

The set of essential nodes $\mathcal{I}_0$ is given along with the datasets; $\mathcal{I}_P$ is obtained by taking a certain percentile of the most highly ranked nodes. The issue of ties is dealt with in the following way.

In looking at the intersection of both sets we get the number of correctly identified nodes. We then normalize that number by the total number of nodes in $\mathcal{I}_P$ as to get the *fraction* of essential nodes among the "thought–to–be–important" nodes — and also to tackle the effect of including the ties. So the higher the value $|\mathcal{I}_O \cap \mathcal{I}_P|/|\mathcal{I}_P|$, the better the ranking scheme performs.

The general benchmark for the different ranking schemes is the global fraction of essential nodes in each network. This threshold lies around 20% of the nodes in each graph, so if one randomly picks, say, 5% of the nodes, then roughly every fifth node should be essential.

[7] Dropping the notion of perturbation for the moment.

It is clear that any ranking scheme with a "success" ratio below this threshold is rather useless, as (statistically) one would be better off to just randomly pick nodes instead of going through the trouble of applying that scheme.

Presentation of the results

We will present the results from this section in Table 5.1 below. Each entry corresponds to the percentage of essential nodes among respectively the top 1%, 5%, 10% or 25% of nodes from the different rankings.

The best performance(s) in each column are marked in a **bold font**. Ratios falling below the threshold of random picks are ~~canceled~~. The question marks in the row corresponding to the ranking by damage value mean that the value has not been calculated due to the numerical effort needed in order to determine it.

In addition to our usual set of ranking schemes we also combined some of them to see if their "synergy" gives improved performance. These combinations are called I_a, I_d and I_c and correspond to combinations of **a**ll[8], the **d**egree–related[9] and the **c**entrality–related[10] rankings. By combination we mean the intersection of the sets of nodes identified as important by each one of them. So to say they represent the "general agreement" of the member schemes.

As an example, the **50.0** entry in the I_a row, Uetz dataset, 5% column, means that if we take a fixed top fraction of nodes from all rankings such that their intersection contains about 28 nodes (that is 5% of the 558 nodes in the largest connected component of the dataset), then every second node in that set is, in fact, essential.

With these remarks made, let's take a look at Table 5.1.

Rank. sch.	Uetz dataset				Ito dataset				Yu dataset			
	1%	5%	10%	25%	1%	5%	10%	25%	1%	5%	10%	25%
ND	**83.3**	48.4	34.4	31.5	28.1	23.8	**21.6**	**19.9**	54.3	42.3	48.1	**41.6**
HITS	50.0	35.7	28.6	27.2	~~17.1~~	~~16.6~~	~~15.4~~	~~16.5~~	22.7	43.8	48.6	35.9
PR	**83.3**	46.4	39.3	**32.6**	28.1	**24.8**	**21.6**	19.1	58.0	62.1	54.0	39.8
Exc.	39.1	24.6	24.6	24.8	18.4	18.4	18.4	18.4	34.7	34.7	34.7	23.5
Stat.	33.3	~~17.2~~	26.8	26.8	**31.3**	19.7	18.0	~~16.9~~	56.0	59.5	52.0	33.6
CV	33.3	~~17.2~~	26.8	26.8	**31.3**	19.7	18.0	~~16.9~~	56.0	59.5	52.0	33.6
Dam.	66.7	44.0	33.9	30.6	?	?	?	?	?	?	?	?
I_a	65.7	**50.0**	**41.1**	30.3	28.1	19.3	~~17.4~~	18.3	**80.4**	68.4	51.8	39.7
I_d	**83.3**	48.3	37.5	29.7	30.0	19.0	~~17.3~~	~~17.9~~	79.2	**72.2**	**54.2**	40.6
I_c	50.0	~~17.2~~	28.6	26.8	**31.3**	19.7	~~17.4~~	~~17.9~~	55.3	60.3	46.2	33.6

Table 5.1: *Comparison of the ranking schemes abilities to identify different quantities of essential nodes in the different datasets. The success rates for random picks are 22.6%, 17.9% and 21.0% for the respective datasets.*

[8] Including damage, when available
[9] Node degrees, PageRank and HITS
[10] Excentricity, status and centroid value

5.3.2 Uetz dataset

The results for this dataset look quite convincing (with the exception of the drop in performance of the status and centroid value induced rankings at 5%) and in a way intuitive: Generally, all ranking schemes decrease in performance as we ask for more and more important nodes. This makes sense, because the more nodes we demand, the more likely it will be that ranking scheme and reality "disagree"; whereas if we only want a handful of the most important nodes, the vast majority of them should (hopefully) be essential.

In this dataset, the global fraction of essential nodes is 22.6%. When we ask only for a few nodes, node degrees and PageRank share the first place, together with I_d (their combination with HITS). Demanding for larger numbers of important nodes, it is the combination of all rankings schemes I_a that has the highest success rate, about one in two are correctly identified. If we consider an even larger fraction, a quarter of the nodes, about one third of the nodes identified by PageRank are essential.

In any case, status, centroid value and excentricity perform worst, whereas in contrast the combination of all of the schemes gives consistently good results. In a way surprising, damage also performs quite well, at least better than any of the centrality based ranking schemes.

5.3.3 Ito dataset

The situation is quite different in this dataset, as excentricity, status and centroid value perform better relative to PageRank and node degrees than in the previous case. However, the latter two again allow for the highest success rates when we ask for more than 140 proteins (5% of the network size).

The benchmark of 17.9% in case of random picks is however crossed a number of times — and, surprisingly, HITS lies *entirely* below it. In case of desired 10 and 25% of nodes, status and centroid value are not doing well (contrary to their good performance for the very top ranked 1% of nodes).

Probably due to their low fraction of essential nodes, all the combination methods fail as well to deliver better results than just randomly picking nodes.

Excentricity performs also quite bad, and the constant values due to the fact, again, that the ranking induced by the excentricities of the nodes is extremely clustered.

5.3.4 Yu dataset

The first thing we would like to point out is that we can confirm the result from the paper by Yu *et al.* [64]. They defined 1061 nodes (around 22% of the total network size) of the most highly connected nodes as hubs, and found that about 43% of them are essential. In taking about 25% of the nodes with the highest degrees this fraction is has decreased slightly to about 41.6%. This is in agreement with their prediction in the supplementary material to their paper [65].

Containing 953 essential nodes, about 21% of randomly picked nodes (from the largest connected component) should be essential. It is positive to see that, unlike the Ito dataset, this threshold is not crossed here.

Most notable is, however, that again the combination of all ranking schemes or just the degree based schemes provide best results in this dataset. However, status and centroid value also allow for more than a third of correctly identified nodes.

Strangely again, HITS gives a very bad result when we look at the very top ranked nodes, but then increases to come close to a 50% success rate.

5.3.5 Comparison between the datasets

In general, all ranking schemes work best on the Uetz dataset, correctly identifying at least one fifth of nodes (except for the one case of centroid value and status). We also find the highest ratio, that is 83.3% of essential nodes among the top scoring nodes with respect to node degrees and PageRank.

Good results are also obtained for the large dataset by Yu *et al.*. With two small exceptions, at least a third of the nodes are correctly identified, and in the majority of cases even more than 50%.

Maybe due to the quality of the data and lacking robustness in the ranking schemes, they perform not as well on the Ito dataset. Here, at most a third of the identified nodes are essential. Some rankings schemes even drop below the threshold where one would be better off to just randomly pick nodes.

Among the different schemes, it is again the centrality based ones that perform worst. Damage however seems to work quite well, at least on the Uetz dataset. Further investigations might show whether this will also be the case for other dataset. It is doubtful though that the quality of the results will be able to justify the disproportionately higher computational effort.

Having examined the ability of the different ranking schemes to identify essential nodes in the datasets, we would like to close this last chapter by testing the *robustness* of this identification process.

5.4 Robustness of the identification of essentiality

Some of the strange effects from the previous section, like the sudden but "temporary" collapse of the values from centroid value and status at 5% in the Uetz dataset, can surely be explained by the considerable amount of noise in the datasets.

To further investigate the effect of perturbations on the rankings we made another series of simulations.

5.4.1 Our setup

We introduced different amounts (5%, 10%, 15% and 20% of the number of edges) of edge perturbations (removal, rewiring and addition) to each of the datasets and averaged over 50 runs.

The results are presented in a similar manner as in the previous section, with the important difference that the percentages in the table head now correspond to the amount of perturbation introduced. As in the previous chapter for

deviation measures 4 and 5 we consider the top 5% of the nodes to be important, and then determine the fraction of nodes that are know to be essential.

Again, the best values in each row are highlighted in **bold font** and values that fall below the threshold of random picks are ~~canceled~~. The first column recalls the performance of the ranking scheme on the unperturbed network.

5.4.2 Uetz dataset

The results for our first dataset are shown in Table 5.2 below.

Rank. sch.	No pert.	Edge removal				Edge rewiring				Edge addition			
		5%	10%	15%	20%	5%	10%	15%	20%	5%	10%	15%	20%
ND	**48.4**	**48.6**	**45.3**	**46.3**	**47.1**	**48.3**	**47.9**	**47.1**	46.3	**47.6**	**46.5**	**46.1**	45.7
HITS	35.7	35.2	35.9	36.8	37.2	37.2	38.9	40.1	39.7	37.1	38.9	40.2	41.9
PR	46.4	44.6	44.6	44.4	45.2	45.6	46.7	46.6	**47.0**	45.4	45.2	45.1	**46.4**
Exc.	24.6	~~20.6~~	~~18.8~~	~~19.1~~	~~16.8~~	26.3	22.7	~~20.4~~	~~19.6~~	32.6	29.8	27.9	27.4
Stat.	~~17.2~~	~~19.1~~	~~18.9~~	~~19.1~~	~~16.8~~	23.2	~~21.7~~	~~20.4~~	~~19.4~~	~~19.7~~	26.1	28.3	29.7
CV	~~17.2~~	~~19.1~~	~~18.9~~	~~19.1~~	~~16.8~~	23.2	~~21.7~~	~~20.4~~	~~19.4~~	~~19.7~~	26.1	28.3	29.7
Dam.	44.0	41.4	42.1	41.4	40.6	37.0	37.0	37.2	36.2	37.6	35.7	35.0	34.5

Table 5.2: *Comparison of the ranking schemes' abilities to identify essential nodes under three types of perturbation, Uetz dataset.*

Edge removal

One thing to be noticed immediately is that the node degree induced ranking is clearly the most successful ranking scheme here, closely followed by PageRank. It is quite impressive to see that even with up to 20% of the edges removed still about 47.1% of the identified proteins are essential to the survival of the organism.

On the other hand, the rankings from the centrality based measures are again the most sensitive ones, and their "performance" lies even below that of random picks.

The slightly strange increase of the fraction of essential nodes in the HITS rankings should not be mistaken as an increase of quality. We should rather consider the absolute difference of the fractions, and with a change of at most 1.5% HITS seems to be the most robust of the schemes.

Damage to the contrary appears to be quite sensitive, dropping up to 3.4% from 44.0% down to 40.6%. However, it still allows for higher detection rates than HITS for instance.

Edge rewiring

Again, node degrees can hold their leading position with the exception of PageRank which seems to deliver better results at 20% of rewired edges.

This should be reconsidered, as we witness a strange *increase* in quality (also the case with HITS). The centrality based measure show a "leap" forward, the detection rate changing up to 6% with only 5% of the edges rewired. After that, they fall below the threshold of random picks again.

The most robust measure here is PageRank, as it only deviates by at most 0.8% from the unperturbed result.

Edge addition

Looking at the columns of edge addition we find quite similar results as for edge rewiring.

The most sensitive measures here again are clearly the centrality based ranking schemes, status and centroid value deviating by as much as 12.5% — strangely enough in the "good" direction though.

In contrast to that stands PageRank, being the most robust ranking scheme here again. Node degrees have the highest fractions of correctly identified nodes.

5.4.3 Ito dataset

The data obtained from perturbations of this dataset are displayed in Table 5.3 below.

Rank. sch.	No pert.	Edge removal				Edge rewiring				Edge addition			
		5%	10%	15%	20%	5%	10%	15%	20%	5%	10%	15%	20%
ND	23.8	24.0	**24.1**	**24.4**	**24.2**	23.6	23.6	23.4	23.0	24.1	**25.3**	**24.9**	**24.4**
HITS	~~16.6~~	~~16.6~~	~~16.7~~	~~15.9~~	~~15.5~~	~~16.8~~	~~16.7~~	~~16.8~~	~~17.0~~	~~16.4~~	~~16.2~~	~~15.7~~	~~15.8~~
PR	**24.8**	**24.6**	**24.1**	**24.4**	23.9	**24.5**	**24.2**	**24.2**	**23.8**	**24.3**	23.8	23.5	23.4
Exc.	18.4	18.0	18.1	~~17.4~~	~~17.5~~	~~17.9~~	~~17.8~~	18.6	18.2	18.1	~~17.6~~	~~17.9~~	~~17.6~~
Stat.	19.7	19.9	~~17.7~~	~~17.2~~	~~17.5~~	19.7	20.3	20.7	20.3	18.9	18.7	18.7	18.7
CV	19.7	19.9	~~17.7~~	~~17.2~~	~~17.5~~	19.7	20.3	20.7	20.3	18.9	18.7	18.7	18.7

Table 5.3: *Comparison of the ranking schemes' abilities to identify essential nodes under three types of perturbation, Ito dataset.*

Edge removal

In this setting, node degrees and PageRank share the best results being at least 5 percentage points more successful than all the other measures.

Not only HITS, which always seems to detect less essential nodes than one would get by just randomly picking nodes, but also the centrality based ranking schemes give poor results.

The most robust of all schemes is node degrees, deviating by up to 0.6% compared to the success rate on the unperturbed network.

Edge rewiring

When it comes to edge rewiring, the centrality based ranking schemes as well as HITS appear again to be increasingly successful. Most interesting however is the fact that they are the most robust here, only deviating by up to 0.6%.

PageRank on the other hand has clearly the highest fraction of essential nodes, followed by node degrees.

Edge addition

Here, PageRank can only hold the first place for up to 5% added edges, as node degrees tops it soon after.

Centroid value and status show an interesting behaviour: the success rate remains on a constant level after about 5% of edges have been added, signaling very good robustness to this type of perturbation.

As for all perturbations, HITS never makes it above the threshold of random picks, disqualifying it in this dataset.

5.4.4 Yu dataset

See Table 5.4 below for the results from our sensitivity analysis on this dataset.

Rank. sch.	No pert.	Edge removal				Edge rewiring				Edge addition			
		5%	10%	15%	20%	5%	10%	15%	20%	5%	10%	15%	20%
ND	42.3	42.7	42.8	43.0	43.2	43.1	43.0	43.2	43.3	42.3	42.3	42.3	42.4
HITS	43.8	44.6	44.5	44.6	44.6	44.6	44.5	44.6	44.5	43.8	43.8	43.8	43.8
PR	**62.1**	**61.7**	**61.5**	**61.3**	**61.0**	**62.8**	**63.4**	**64.0**	**64.1**	**62.0**	**62.5**	**62.9**	**63.2**
Exc.	34.7	34.3	36.3	35.7	34.2	34.0	32.9	31.6	29.2	31.8	30.2	30.1	28.0
Stat.	59.5	56.6	53.2	51.1	48.1	58.7	58.8	57.9	57.1	59.7	59.7	59.8	59.8
CV	59.5	56.6	53.2	51.1	48.1	58.7	58.8	57.9	57.1	59.7	59.7	59.8	59.8

Table 5.4: *Comparison of the ranking schemes abilities to identify essential nodes under three types of perturbation, Yu dataset.*

As the behaviour of the different ranking schemes is quite similar across all three types of perturbation we shall not split the comment as for the other datasets. For all types of perturbations, PageRank consistently allows for the highest fraction of essential nodes in its top ranked 5% of nodes.

Perhaps surprisingly it is followed by status and centroid value which, in contrast to their performance on the other datasets, now give quite good results and with (but for one exception) continuously more than 50% of correctly identified nodes perform clearly better than even node degrees.

The third "block" form HITS and node degrees with around 44% success rate, and excentricity is, again, the least effective ranking scheme.

When it comes to robustness however, it is HITS which is always the least sensitive to any type of perturbation, in case of edge addition even being completely robust.

Status and centroid value however drop as much as 11.4% in performance when we remove up to 20% of the edges.

Comparison between the datasets

Across the datasets we could see that node degrees and PageRank gave best results, and that in a very consistent manner.

When it comes to robustness, it was HITS that most of the time was the least sensitive to the different types of perturbation. The centrality based measures stayed behind these node degree oriented measures, both in robustness and in the absolute quality of the results.

Damage looked quite promising in the Uetz dataset, and the investigations should definitely be extended to the other datasets.

Having looked at the application of the ranking schemes to some real datasets in this chapter, we now would like to move on to the final conclusion.

Conclusion

In the first chapter of this *Studienarbeit* we introduced basic notions from graph theory and presented methods to quantify various properties of graphs together with MATLAB implementations of these algorithms.

This laid the theoretical basis for the analysis of three major random graph models in the second chapter. We presented the classical Erdös–Rényi random graph model, the small–world model by Watts and Strogatz, as well as the scale–free model by Barabási and Albert. We explained and implemented their generating algorithms, investigated and compared their properties, and discussed examples of their use to simulate real world networks.

Chapter 3 then dealt with ranking schemes. Besides the intuitive use of the degree of a node as a measure of importance we presented two eigenvector based ranking algorithms commonly used in information retrieval systems, namely the HITS and the PageRank algorithms. Their setup and the theory behind their convergence properties was explained.

We also introduced three centrality measures that have been traditionally used to solve resource allocation problems, as well as a more recent measure of importance, damage, which has been suggested for the use in biological networks.

For all of these measures readily deployable MATLAB implementations have been written.

If one intends to apply ranking schemes to real, measured data — which is almost always inaccurate and noisy to some extent — it is vital to investigate their sensitivity to variations in the network. In the fourth chapter, we set up several measures of deviation to quantitatively assess the robustness of the ranking schemes. We also presented a few theoretical results concerning HITS and PageRank, but the bounds given by these theorems proved to be rather conservative and not tight.

The simulations we presented show that while no single ranking scheme is optimal in all situations, nonetheless some reasonably general conclusions can be drawn. For instance, although PageRank and the simpler node degree induced ranking proved many times to be the most robust of the seven ranking schemes, their leading position has been also challenged a number of times by the HITS algorithm as well as damage. On the other hand, the centrality based algorithm, especially excentricity, seemed to be the most sensitive measures of importance.

The choice of the ranking scheme of course depends not only on its robustness. Other factors would be the resolution of the ranking (excentricity

for instance produced highly coarse and clustered rankings), and, more importantly, the actual *interpretation* of the importance a ranking scheme attributes to its most highly ranked nodes. It is surely the relation between importance by hight ranking and importance in the context of the particular application being considered that will favour one scheme or the other.

For that reason, the fifth chapter looked at one possible application of the ranking schemes. We used three datasets of protein–protein interaction networks that have recently become available to investigate the connection between ranked importance and essentiality (a node being essential if its removal would cause the organism to die).

We found that, depending on the dataset and the top fraction of nodes one actually considers to be important, the best results are obtained with node degrees and PageRank, or combinations of both, allowing in one case for more than 80% of correctly identified nodes. These interesting results certainly motivate further and more extensive studies of those ranking schemes as well as combinations of them.

To extend our analysis of robustness from the fourth chapter, we also introduced increasing amounts of perturbation to the datasets. It was HITS that seemed to be the most robust, in that it seemed to be identifying the same amount of essential nodes independent of the amount of perturbation. However, node degrees and PageRank still allowed for the highest success rates.

The most surprising general finding was probably that pure node degrees usually performed very well if not best, whereas their calculation was by far the least costly. This suggests that node degrees may provide a reasonable indication of protein essentiality at very low computational cost.

A number of open questions remain, and others have appeared. On the one hand, the evaluation of robustness could be extended to other graph models and to directed graphs. On the other, one could introduce different measures of deviation that may be more appropriate or related to a certain application, and one could also investigate the effect of more structured perturbations (such as intentional attacks on the most highly connected nodes) or of combinations of those we presented here.

Also, further theoretical research should be able to either refine the results on the robustness of PageRank and HITS (for example by taking into account the actual type of perturbation) or to extend it to other ranking schemes.

Due to the enormous computational effort needed to calculate the damage of nodes, we could not apply the damage ranking to the larger datasets. It would be interesting to evaluate its performance on these networks as well.

Finally, the techniques discussed here should also be applied to other biological datasets, or datasets from other areas.

APPENDIX A

Additional theorems

A.1 Connectedness and irreducibility

The following theorem states the relation between connectedness of a graph and irreducibility of its adjacency matrix:

Theorem A.1 (Connectedness and irreducibility)

A graph $\mathscr{G} = (\mathcal{V}, \mathcal{E})$ is strongly connected if and only if its adjacency matrix $\boldsymbol{A}_{\mathscr{G}}$ is irreducible.

Proof:

$\boxed{\Leftarrow}$ Assume $\boldsymbol{A}_{\mathscr{G}}$ is reducible. Then it can be rearranged into the form

$$\boldsymbol{\Pi}^T \boldsymbol{A} \boldsymbol{\Pi} = \begin{array}{c} \\ \mathcal{S}_1 \\ \mathcal{S}_2 \end{array} \begin{array}{c} \begin{array}{cc} \mathcal{S}_1 & \mathcal{S}_2 \end{array} \\ \begin{bmatrix} \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{0} & \boldsymbol{D} \end{bmatrix} \end{array} \tag{A.1}$$

where $\boldsymbol{B} \in \mathbb{R}^{r \times r}$, $\boldsymbol{D} \in \mathbb{R}^{(n-r)\times(n-r)}$, $\boldsymbol{C} \in \mathbb{R}^{r\times(n-r)}$ and $\boldsymbol{0} \in \mathbb{R}^{(n-r)\times r}$ is the zero matrix.

There, no node in $\mathcal{S}_1$ can be reached from nodes in $\mathcal{S}_2$, otherwise there would be an edge (u, v) with $u > r$ and $v \leq r$. This would imply that $a_{uv} \neq 0$ which is in contradiction to the assumption that $\boldsymbol{A}_{\mathscr{G}}$ is reducible.

$\boxed{\Rightarrow}$ If $\mathscr{G}$ is not strongly connected then there exists at least one pair of nodes u and v such that v cannot be reached from u.

Let $\mathcal{S}_2$ be the set of edges that can be reached from u. Without loss of generality, let $\mathcal{S}_2 = \{r+1, \ldots, n\}$, with $1 \leq r < n$. Then, there cannot be an edge (i, j) with $i \in \mathcal{S}_2$ and $j \notin \mathcal{S}_2$, because j cannot be reached from u. This implies $a_{ij} = 0$ for all $i > r$ and all $j \leq r$ □

As mentioned in Subsection 1.3.1 on page 5, a matrix $\boldsymbol{A}$ is irreducible if and only if there exists a positive integer m such that $(\mathbf{id} + \boldsymbol{A})^m \succ \boldsymbol{0}$.

A.2 Perron–Frobenius Theorem

We now state one formulation of the famous 1907 theorem of the German mathematician Oskar Perron:

Theorem A.2 (Perron's Theorem)

If $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{A} \succ \boldsymbol{0}$ then there exists $\lambda_1 > 0$ and $\boldsymbol{\nu}_1 \succ \boldsymbol{0}$ such that the following hold:

(i) $A\boldsymbol{\nu}_1 = \lambda_1 \boldsymbol{\nu}_1$,

(ii) if $\lambda \neq \lambda_1$ *is any other eigenvalue of* $\boldsymbol{A}$, *then* $|\lambda| < \lambda_1$,

(iii) λ_1 *is an eigenvalue of geometric and algebraic multiplicity 1.*

Proof: Can be found for example in [42]. □

Due to the work of another German mathematician, Ferdinand Georg Frobenius, this result was five years later extended to the following theorem:

Theorem A.3 (Frobenius' Theorem)

If $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{A} \succeq \boldsymbol{0}$ and $\boldsymbol{A}^m \succ \boldsymbol{0}$ for some positive integer m then conclusions (i)–(iii) of Theorem A.2 apply to $\boldsymbol{A}$.

Proof: Again, can be found for example in [42]. □

As the two theorems go hand in hand, if we refer in the above document to the "Perron–Frobenius Theorem", we refer to the combination of both Perron's and Frobenius' Theorems.

APPENDIX B

Additional plots

B.1 Sensitivity

The following plots have been moved here so as not to further clutter up the fourth Chapter. The major difference is that here the sensitivity of the ranking induced by the concept of damage is also included.

B.1.1 General remarks

The setup

The generating parameters for the graphs used for the following plots are basically the same as above, with the difference that size is scaled down 20 times. So we have $n = 150$ nodes, but still $n_0 = m = 3$, resulting in roughly 450 edges (444 to be precise). The perturbations go as far as to remove, add or rewire 100 edges, or remove 20 nodes.

In Chapter 4, we removed up to 100 nodes, which would correspond to removing up to 5 nodes in the smaller graphs. So it is clear that removing up to 20 nodes has a much higher impact, and disconnection here does become an issue (recall that this was not the case for the larger graphs).

Every value from the plots corresponds to the average value from 250 independent runs. As above, the following markers have been used for the different ranking schemes:

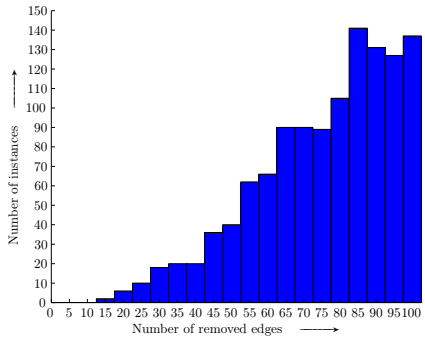
- × Node degrees
- ○ HITS
- □ PageRank
- △ Excentricity
- ◊ Status
- • Centroid value
- ✳ Damage

Disconnection

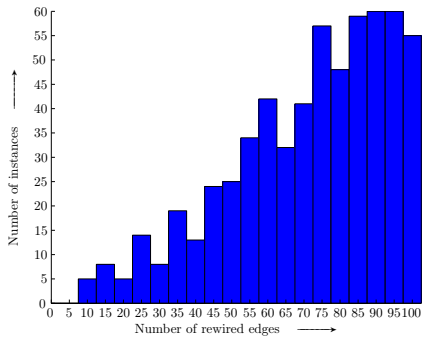
As for the investigations in the fourth Chapter, we should first take a look at the point where most of the graphs get disconnected.

The corresponding plots in Figure B.1 on the next page for edge removal and rewiring roughly imitate the shape of their counterparts from Figure 4.1 on page 67, more precisely the first "half" of those plots, i. e. up to 1000 perturbed edges. It is important to mention that a number of graphs did not disconnect in the process (this fraction is not shown in the plots).

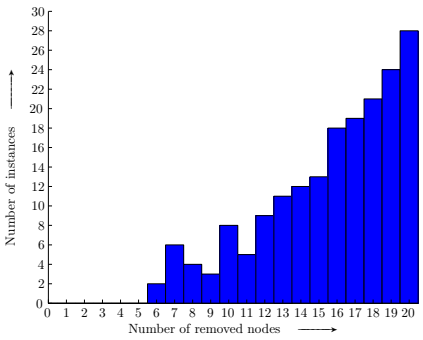
For that matter, Table B.1 on the next page shows the percentage of graphs that stayed connected even after maximum perturbation of 200 removed or rewired edges, or 20 removed nodes (in the "DND" row). The listed mean



(a) Edge removal



(b) Edge rewiring



(c) Node removal

Figure B.1: *Distribution of disconnected graphs for different amounts of perturbations. A total of 2250 resp. 1875 graphs were used for the edge resp. node perturbations.*

values (accompanied by their respective standard deviations) are the mean values of the number of removed/rewired edges/nodes that disconnected the graph (ignoring the graphs that did not get disconnected).

Value	**Perturbation**		
	E. removed	E. rewired	N. removed
DND (%)	47.11	72.93	90.24
mean	75.79	71.33	15.66
st. dev.	19.09	22.13	3.75

Table B.1: *Comparison of the disconnection characteristics of the different perturbations. "DND" stands for "did not disconnect", corresponding to the fraction of graphs that did not get disconnected after the perturbations of up to 200 removed or rewired edges, or 20 removed nodes. Mean value together with standard deviation correspond to the mean number of perturbed edges (nodes) needed to disconnect the graph.*

Our additional plot for node removal, Figure B.1(c), is also in accordance with the observations from Chapter 4: until up to five removed nodes the small graphs never disconnected. Scale this number by the factor of 20, and you find the corresponding threshold of 100 nodes for the big graphs that we mentioned on page 67.

The effects of the graphs becoming disconnected on the ranking schemes are mainly, as described earlier, that excentricity as a means for ranking becomes useless, as all nodes will get the same rank allowing for no distinction to be made between nodes.

Distribution of scores

To get an idea of the "resolution" of the rankings, we present the distributions of scores of the different importance measures (again averaged out and sorted in increasing order) for 200 unperturbed graphs in Figure B.2. As mentioned

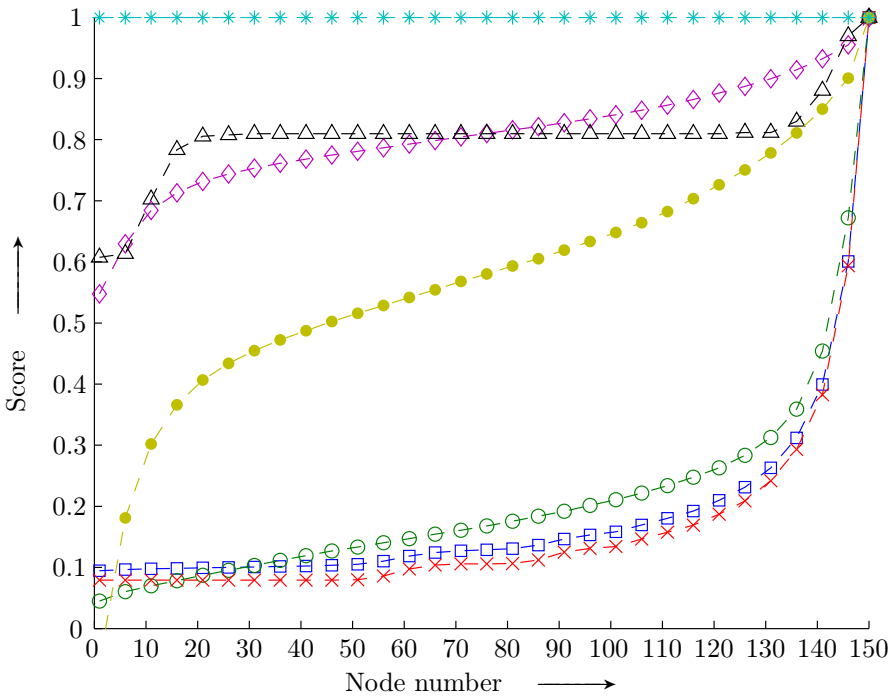


Figure B.2: *Average distribution of the scores of the different ranking schemes on 200 Barabási–Albert graphs with $n = 150$ nodes and $n_0 = m = 3$.*

earlier, damage is not very useful a measure when graphs are as robustly connected as is the case with ours (which again are designed to closely resemble real world networks). Excentricity scores are also very coarse, indicated by the broad intervals of same score.

In the current situation, PageRank, node degrees and HITS seem to be highly correlated. Their pronounced rising allow for good accuracy in discrimination of important nodes.

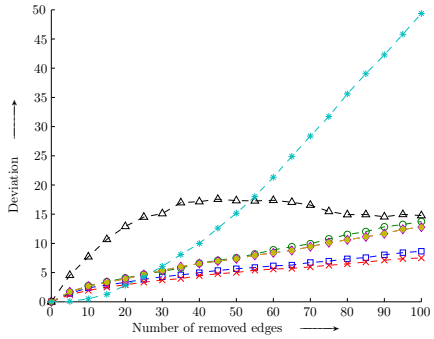
B.1.2 Results

Method 1

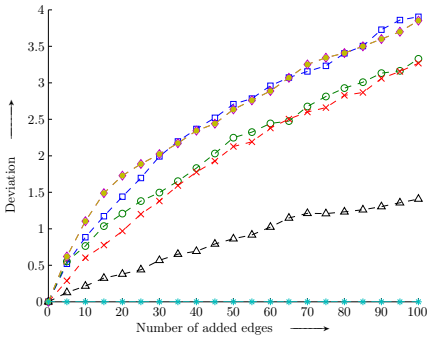
Most of the curves in Figure B.3 resemble the shape of their counterparts for the larger graphs (Figure 4.3 on page 69), more precisely the sections of up to 1000 perturbed edges.

From about 25 removed edges onward, damage's sensitivity seems to increase linearly with a much steeper slope than that of HITS, PageRank, status and centroid value. In case of edge addition, it does not deviate at all, as all nodes have and keep the same score.

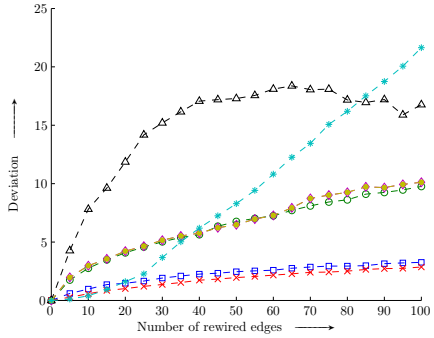
PageRank and node degrees are only outperformed once, that is by excentricity in the case of edge addition. This must not be taken too seriously in light of the coarseness of excentricity induced rankings.



(a) Edge removal



(b) Edge addition



(c) Edge rewiring

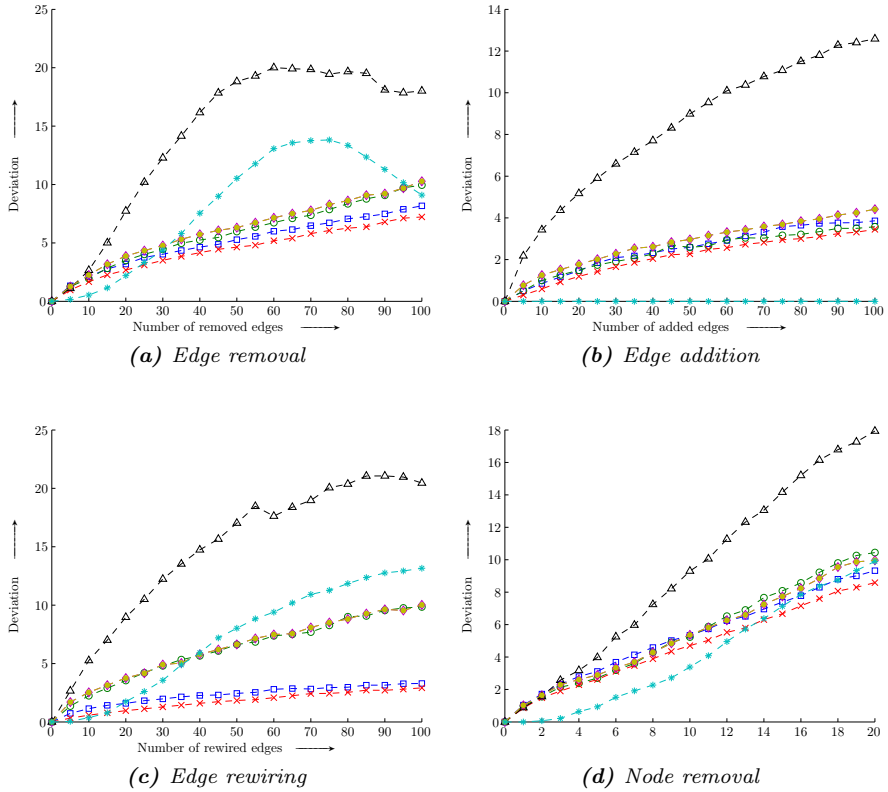
Figure B.3: *Sensitivity for deviation measure 1.*

Method 2

Again, we find strong similarity between the results from the larger graphs and the ones presented in Figure B.4.

Interestingly enough, however, the sensitivity of damage seems to decrease again after a removal of about 70 edges. Looking at edge rewiring and node removal, it seems to be roughly as robust as the other measures.

PageRank and node degrees show the least deviations for all four types of perturbations, node degrees being slightly better than PageRank.



(a) *Edge removal*

(b) *Edge addition*

(c) *Edge rewiring*

(d) *Node removal*

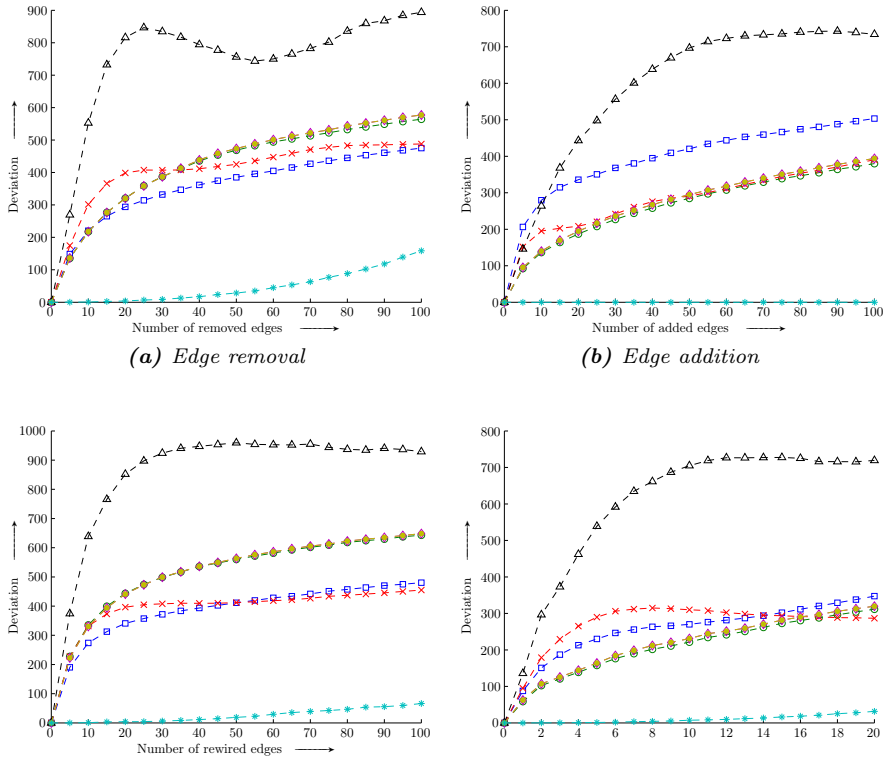
Figure B.4: *Sensitivity for deviation measure 2.*

Method 3

Looking at Figure B.5 the only surprising thing to notice is the slight oscillations in sensitivity of node degree induced rankings, very similar to the effects observed in Figure 4.5 on page 72.

It is interesting to see that damage seems to be very robust with respect to maximum deviations in the rankings.

PageRank and node degree here are outperformed by HITS, status and centroid value in the cases of edge addition and node removal, as we already observed in Chapter 4.



(a) Edge removal *(b) Edge addition*

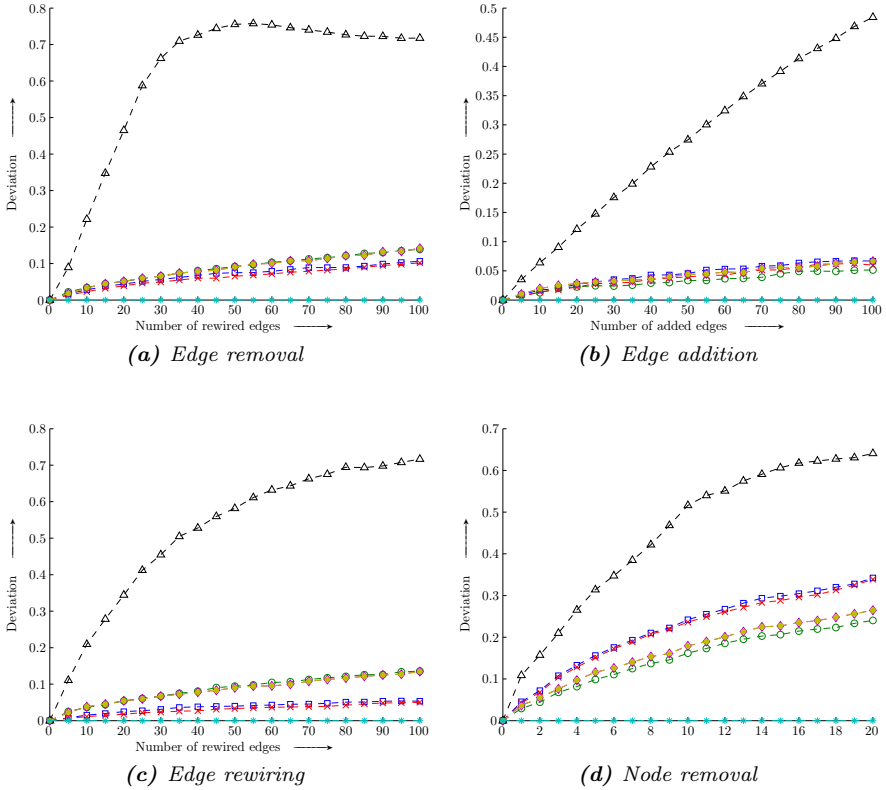
(c) Edge rewiring *(d) Node removal*

Figure B.5: *Sensitivity for deviation measure 3.*

Method 4

Besides many similarities between Figure B.6 and its counterpart on page 73, the plot for edge removal is strikingly different, especially in that the performance of the different ranking schemes seems to be inversed: if PageRank and node degrees have been very robust, they are now clearly outperformed by centroid value, status and HITS — whereas HITS seemed to be the most sensitive measure of importance in Figure 4.6(d) on page 73 now is the most robust.

Damage however has a flawless appearance for all four types of perturbation, which can be explained easily. At the beginning (no perturbation) all the nodes are important (all of them share the first place, as the graph is connected and will stay connect no matter which node is removed), so there is no chance that important nodes of the perturbed graph have not been important in the first place.



(a) *Edge removal*

(b) *Edge addition*

(c) *Edge rewiring*

(d) *Node removal*

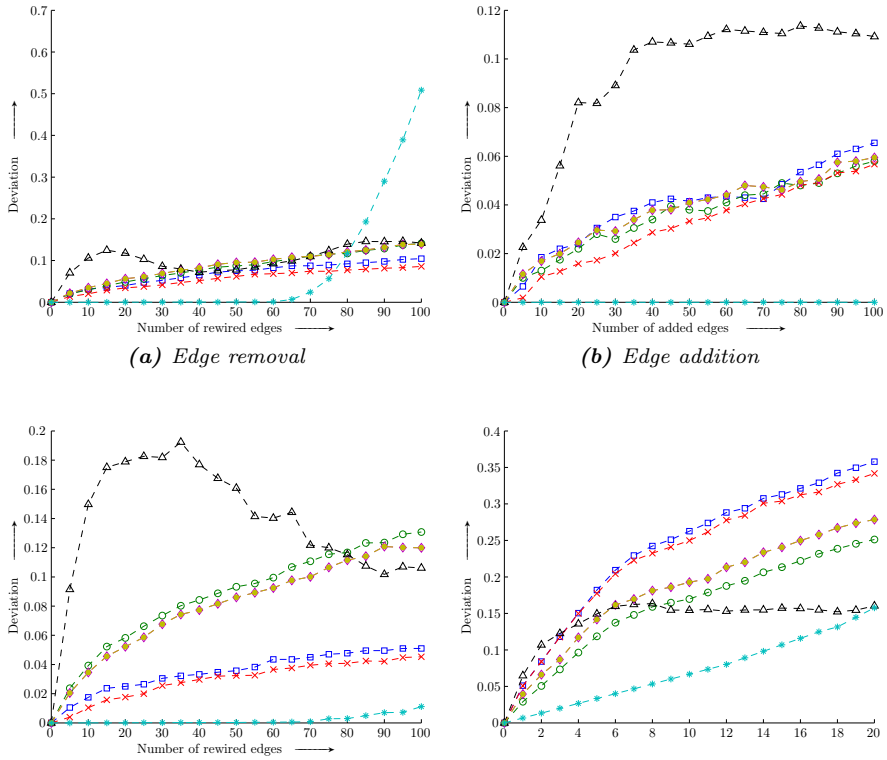
Figure B.6: *Sensitivity for deviation measure 4.*

Method 5

Figure B.7 brings few surprises, as it again is very close to its companion (Figure 4.7 on page 75).

Damage, in the case of edge removal, shows a late but pronounced rise in sensitivity, being rather robust up to about 60 removed edges. When it comes to edge rewiring or node removal it is much more stable however and outperforms the other measures in that regard.

Otherwise, PageRank and node degrees are reasonably robust as well, but are beaten by HITS, centroid value and status in the case of node removal.



(a) Edge removal *(b) Edge addition*

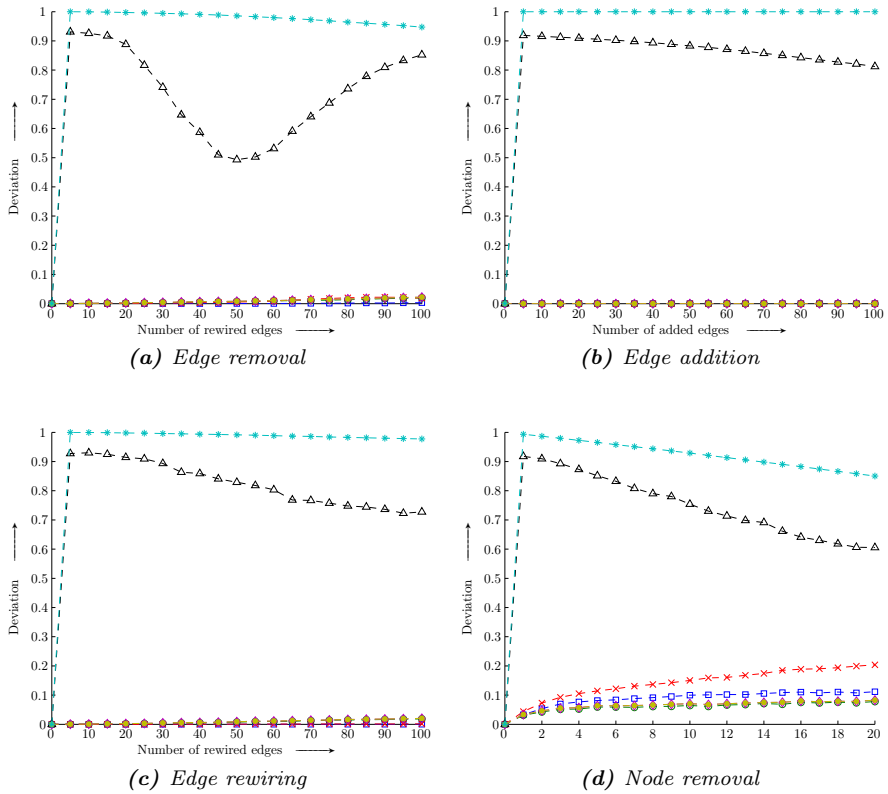
(c) Edge rewiring *(d) Node removal*

Figure B.7: *Sensitivity for deviation measure 5.*

Method 6

Comparison of the last plot of this series, Figure B.8, with the corresponding plot from Chapter 4 (on page 76) also show reasonable amounts of resemblance. Excentricity is way out of scope, so is damage. This can be easily explained by the combination of poor ranking and the way the deviation measure is set up (all nodes get first place, which in that case, curiously, has as consequence that all nodes end up in the "loser set" $\mathcal{L}$).

The other measures however perform very well, especially in case of edge addition.



(a) *Edge removal* ***(b)*** *Edge addition*

(c) *Edge rewiring* ***(d)*** *Node removal*

Figure B.8: *Sensitivity for deviation measure 6.*

B.1.3 Conclusions

We have seen that the plots from this series of simulation resemble quite closely to their bigger counterparts from the 20 times larger networks.

Here as well, PageRank and node degree induced rankings show most of the time the least sensitivity to perturbations, being challenged however by HITS, centroid value and status in some cases of edge addition or node removal.

The curves for excentricity should not be given too much weight, as, again, on the one hand the induced ranking is extremely coarse and on the other the impact of disconnection of the graph is significant.

List of Symbols

General notations

n	Scalars; lowercase letters[1]
$\boldsymbol{x}$	Vectors; lowercase bold letters (always supposed to be column vectors, if not transposed), or elementwise in parentheses: $\begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix}^T$
$\boldsymbol{M}$	Matrices; uppercase bold letters, or elementwise in brackets: $\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$
m_{ij}	Element (i, j) of Matrix $\boldsymbol{M}$; lowercase letters with indices
m_{ij}^r	Element (i, j) of matrix $\boldsymbol{M}^r$ (as opposed to $(m_{ij})^r$)
$\mathcal{S}$	Sets; uppercase "calligraphic" letters
$\mathcal{S}^r$	r times the Cartesian product of $\mathcal{S}$, i. e. $\mathcal{S} \times \mathcal{S} \times \ldots \times \mathcal{S}$ exactly r times
$\lvert\mathcal{S}\rvert$	Cardinality of set $\mathcal{S}$
$\mathscr{G}$	Graphs; uppercase "script" letters
$\mathfrak{P}$	Properties, assumptions; uppercase "Fraktur" letters
$\succ$	$\boldsymbol{M} \succ \boldsymbol{0}$ denotes that $\boldsymbol{M}$ is strictly positive, i. e. $m_{ij} > 0$ for all i, j
$\succeq$	$\boldsymbol{M} \succeq \boldsymbol{0}$ denotes that $\boldsymbol{M}$ is strictly non–negative, i. e. $m_{ij} \geq 0$ for all i, j and for at least one element $m_{ij} > 0$ (so $\boldsymbol{M} \neq \boldsymbol{0}$)
$\langle\cdot\rangle$	Usually denotes an average value
$\cdot$	Within matrices stands for a zero entry (for better legibility)

Specific notations

$\boldsymbol{1}_n$	The n–element column vector with all ones, p. 6
$\boldsymbol{1}_{n\times n}$	The $n \times n$ matrix with all ones, p. 49
$\boldsymbol{A}_{\mathscr{G}}$	or just $\boldsymbol{A}$ is usually the adjacency matrix of a graph, p. 4
C_k^n	Binomial coefficient, $\mathrm{C}_n^k = \binom{n}{k} = \frac{n!}{k!(n-k)!}$ for $n \geq k \geq 0$

[1] however, v usually denotes some vertex of a graph

c	Usually the clustering coefficient of a node, p. 14
$z(v)$	Centroid value of vertex v, p. 54
$\langle c \rangle$	(Average) clustering coefficient of a graph, p. 14
δ_*	Measure of deviation in rankings ($* = 1$, 2, 3, 4, 5 or 6), p. 62
$\boldsymbol{\Delta}$	The Distance matrix of a graph, p. 7
d	Usually the diameter of a graph, p. 12
$\eth(v)$	Damage of vertex v, p. 55
$\boldsymbol{e}_i$	i–th unit vector in $\mathbb{R}^n$ (dimensions from context)
$\mathcal{E}$	Edge set of a graph, p. 1
e	The total number of edges of a graph, or sometimes just some particular edge
$e(v)$	Excentricity of vertex v, p. 52
γ	The Euler–Mascheroni constant, $\gamma \approx 0.577\,215\,664\,902$ is the limit of the sequence $\gamma = \lim_{n\to\infty} \left(\sum_{k=1}^{n} \frac{1}{k} - \ln n\right)$, p. 20
$\mathbf{id}_n$	Identity matrix in $\mathbb{R}^{n\times n}$ (dimensions from context)
k	Usually the degree of a node, p. 3
$\langle k \rangle$	Average node degree of a graph, p. 9
l_{ij}	Distance (i. e. length of the shortest path) between nodes i and j, p. 3
$\boldsymbol{L}$	The Laplacian matrix of a graph, p. 6
λ	Usually an eigenvalue of a matrix
$\langle l \rangle$	Average path length of a graph, p. 10
$\boldsymbol{\nu}$	Usually an eigenvector of a matrix
n	Usually the total number of nodes of a graph
$o(\cdot)$	The Landau symbol: $f(m) = o\big(g(m)\big)$ implies $\left\lvert \frac{f(m)}{g(m)} \right\rvert \to 0$ as $m \to \infty$
$\boldsymbol{\pi}^T$	The vector with the PageRank scores of a graph, p. 47
$\bar{\boldsymbol{P}}$	The transition matrix of a graph, p. 48
$\boldsymbol{\Pi}_n$	A permutation of $\mathbf{id}_n$ (dimensions from context): $\boldsymbol{\Pi}_n = \big(\boldsymbol{e}_{s_1} \quad \dots \quad \boldsymbol{e}_{s_n}\big)$, where $(s_1, \dots, s_n)$ is a permutation of $(1, \dots, n)$
p	Usually some probability
$s(\cdot)$	Status of vertex or a graph, p. 53
$\mathcal{V}$	Vertex set of a graph, p. 1

Bibliography

We tried to provide as many publicly accessible web links as possible.[1]

[1] William Aiello, Fan R. K. Chung, and Linyuan Lu. A random graph model for massive graphs. In *STOC '00: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 171–180, New York, NY, USA, May 2000. ACM Press. http://www.math.sc.edu/~lu/papers/random.pdf

[2] Réka Z. Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–96, January 2002. arXiv:cond-mat/0106096

[3] Luís A. Numes Amaral, Antonio Scala, Marc Barthélémy, and H. Eugene Stanley. Classes of small–world networks. *Proceedings of the National Academy of Sciences*, 97(21):11149–11152, October 2000. arXiv:cond-mat/0001458

[4] Albert-László Barabási and Réka Z. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999. arXiv:cond-mat/9910332

[5] Albert-László Barabási, Hawoong Jeong, Zoltan Néda András Schubert, and Tamés Vicsek. Evolution of the social network of scientific collaborations. *Physica A*, 311(3–4):590–614, August 2002. arXiv:cond-mat/0104162

[6] Alain Barrat and Martin Weigt. On the properties of small–world network models. *European Physical Journal B*, 13(3):547–560, January 2000. arXiv:cond-mat/9903411

[7] Vladimir Batagelj and Andrej Mrvar. Networks / Pajek, Program for Large Network Analysis, September 2005. http://vlado.fmf.uni-lj.si/pub/networks/pajek/

[8] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[1] References of the form arXiv:xxxxxxx/yyyyyyy can be retrieved directly from the arXiv.org e–Print archive via http://arxiv.org/abs/xxxxxxx/?yyyyyyy or one of its mirrors.

[9] Norman L. Biggs. *Algebraic graph theory*, volume 67 of *Cambridge Tracts in Mathematics*. Cambridge University Press, London, New York, 1974.

[10] Béla Bollobás. Degree sequences of random graphs. *Discrete Mathematics*, 33(1):1–19, January 1981. http://dx.doi.org/10.1016/0012-365X(81)90253-3

[11] Béla Bollobás. *Random Graphs*. Academic Press, London, UK, 1985.

[12] Béla Bollobás and Oliver Riordan. The diameter of a scale–free random graph. *Combinatorica*, 24(1):5–34, January 2004.
http://www.math.cornell.edu/~durrett/math777/diamSF.pdf

[13] Sergey Brin and Lawrence Page. The anatomy of a large–s1cale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, April 1998. http://www-db.stanford.edu/pub/papers/google.pdf

[14] Andrei Z. Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet L. Wiener. Graph structure in the web. *Computer Networks*, 33(1–6):309–320, 2000. http://www9.org/w9cdrom/160/160.html

[15] Fan Chung and Linyuan Lu. The diameter of sparse random graphs. *Advances in Applied Mathematics*, 26(4):257–279, May 2001.
http://www.math.ucsd.edu/~fan/dia.pdf

[16] Reuven Cohen and Shlomo Havlin. Scale–free networks are ultrasmall. *Physical Review Letters*, 90(5):058701/1–4, February 2003.
arXiv:cond-mat/0205476

[17] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
http://dz-srv1.sub.uni-goettingen.de/sub/digbib/loader?did=D196313

[18] Roger C. Entringer, Douglas E. Jackson, and David A. Snyder. Distance in graphs. *Czechoslovak Mathematical Journal*, 26(2):283–296, June 1976.

[19] Paul Erdös and Alfréd Réyni. On random graphs. *Publicationes Mathematicae*, 6:290–297, 1959.

[20] Ayman Farahat, Thomas Lofaroa, Joel C. Miller, Gregory Rae, and Lesley A. Ward. Existence and uniqueness of ranking vectors for linear link analysis. *SIAM Journal on Scientific Computing*, 2004. Submitted April 2004.

[21] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[22] Agata Fronczak, Piotr Fronczak, and Janusz A. Hołyst. Mean–field theory for clustering coefficients in Barabási–Albert networks. *Physical Revue E*, 68(4):046126/1–4, October 2003. arXiv:cond-mat/0306255

[23] Agata Fronczak, Piotr Fronczak, and Janusz A. Hołyst. Average path length in random networks. *Physical Revue E*, 70(5):056110/1–7, November 2004. arXiv:cond-mat/0212230

[24] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, USA, 2nd edition, 1989.

[25] Google Inc. The Google timeline, December 2004. http://www.google.com/intl/en/corporate/timeline.html

[26] Ramesh Govindan and Hongsuda Tangmunarunkit. Heuristics for internet map discovery. In *IEEE INFOCOM 2000*, volume 3, pages 1371–1380, Tel Aviv, Israel, March 2000. IEEE. http://www.isi.edu/div7/publication_files/heuristics.pdf

[27] Geoffrey Grimmett. *Percolation*. Springer, Berlin, Germany, 2nd edition, 1999.

[28] Frank Harary. Status and contrastatus. *Sociometry*, 22(1):23–43, March 1959. http://links.jstor.org/sici?sici=0038-0431(195903)22:1<23:SAC>

[29] Frank Harary and Robert Z. Norman. The dissimilarity characteristic of husimi trees. *The Annals of Mathematics*, 58(1):134–141, July 1953. http://links.jstor.org/sici?sici=0003-486X(195307)2:58:1<134:TDCOHT>

[30] Taher H. Haveliwala and Sepandar D. Kamvar. The second eigenvalue of the google matrix. Technical Report 2003-20, Stanford University, Department of Computer Science, March 2003. http://dbpubs.stanford.edu/pub/2003-20/

[31] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1985.

[32] Takashi Ito, Kosuke Tashiro, Shigeru Muta, Ritsuko Ozawa, Tomoko Chiba, Mayumi Nishizawa, Kiyoshi Yamamoto, Satoru Kuhara, and Yoshiyuki Sakaki. Toward a protein-protein interaction map of the budding yeast: A comprehensive system to examine two-hybrid interactions in all possible combinations between the yeast proteins. *Proc. of the Nat. Academy of Sciences of the USA*, 97:1143–1147, February 2000. http://biology.uark.edu/ipinto/biol5334/17.pdf

[33] Camille Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869.

[34] Lester R. Ford Jr. Network flow theory. Paper P–923, The RAND Corperation, Santa Moncia, CA, USA, August 1956.

[35] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
http://www.cs.cornell.edu/home/kleinber/auth.pdf

[36] Florian Knorn. Supplementary material to this Studienarbeit, September 2005. http://www.florian-knorn.com/index.php?nav=work&cat=stud

[37] Amy N. Langville and Carl D. Meyer. Deeper inside PageRank. *Internet Mathematics*, 1(3):335–380, 2004.
http://www.internetmathematics.org/volumes/1/3/Langville.pdf

[38] Amy N. Langville and Carl D. Meyer. A survey of eigenvector methods of web information retrieval. *SIAM Review*, 47(1):135–161, March 2005.
http://meyer.math.ncsu.edu/Meyer/PS_Files/Survey.pdf

[39] Ney Lemke, Fabiana Herédita, Cláudia K. Barcellos, Adriana N. dos Reis, and José C. M. Mombach. Essentiality and damage in metabolic networks. *Bioinformatics*, 20(1):115–119, January 2004.
http://www.inf.unisinos.br/~mombach/Ecoli.pdf

[40] Bo Lewin, editor. *Sex i Sverige. Om sexuallivet i Sverige 1996 [Sex in Sweden. About Sex habits in Sweden in 1996].* Statens Folkhälsoinstitut [Swedish National Institute of Public Health], Stockholm, Sweden, 1998.

[41] Fredrik Liljeros, Christofer R. Edling, Luís A. Nunes Amaral, H. Eugene Stanley, and Yvonne Åberg. The web of human sexual contacts. *Nature*, 411(6840):907–908, 2001. arXiv:cond-mat/0106507

[42] David G. Luenberger. *Introduction to Dynamic Systems. Theory, Models & Applications.* John Wiley & Sons, New York, NY, USA, 1979.

[43] Carl D. Meyer. *Matrix analysis and applied linear algebra.* SIAM, Philadelphia, PA, USA, 2000.

[44] Stanley Milgram. The small world problem. *Psychology Today*, 1(2):60–67, May 1967.

[45] Joel C. Miller, Gregory Rae, Fred Schaefer, Lesley A. Ward, Thomas LoFaro, and Ayman Farahat. Modifications of Kleinberg's HITS algorithm using matrix exponentiation and web log records. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 444–445, New York, NY, USA, 2001. ACM Press. http://www.damtp.cam.ac.uk/user/jcm52/pubs/sigir.pdf

[46] Bohan Mohar. Isoperimetric numbers of graphs. *Journal of Combinatorial Theory Series B*, 47(3):274–291, December 1989.
http://dx.doi.org/10.1016/0095-8956(89)90029-4

[47] Bohan Mohar. *Combinatorics, and Applications*, volume 2, pages 871–898. John Wiley and Sons, New York, NY, USA, 1991. Y. Alavi, G. Chartrand, O. R. Ollerman and A. J. Schwenk (ed.).

[48] Cleve Moler. The world's largest matrix computation. MATLAB News and Notes, October 2002.
http://www.mathworks.com/company/newsletters/news_notes/clevescorner/oct02_cleve.html

[49] Mark E. J. Newman, Cris Moore, and Duncan J. Watts. Mean–field solution of the small–world network model. *Physical Review Letters*, 84(14):3201–3204, April 2000. arXiv:cond-mat/9909165

[50] Mark E. J. Newman, Steven H. Strogatz, and Duncan J. Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Revue E*, 64(2):026118/1–17, August 2001. arXiv:cond-mat/0007235

[51] Andrew Y. Ng, Alice X. Zheng, and Micheal I. Jordan. Link analysis, eigenvectors and stability. In *IJCAI '01: Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 903–910, August 2001. http://www.cs.berkeley.edu/~ang/papers/ijcai01-linkanalysis.pdf

[52] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford University, Department of Computer Science, November 1999. http://dbpubs.stanford.edu/pub/1999-66/

[53] Francesco Rao and Amedeo Caflisch. The protein folding network. *Journal of Molecular Biology*, 342(1):299–306, September 2004.
arXiv:q-bio.BM/0403034

[54] Sidney Redner. How popular is your paper? An empirical study of the citation distribution. *European Physical Journal B*, 4(2):131–134, July 1998. arXiv:cond-mat/9804163

[55] Antonio Scala, Luís A. Numes Amaral, and Marc Barthélémy. Small–world networks and the conformation space of a short lattice polymer chain. *Europhysics Letters*, 55(4):594–600, August 2001.
arXiv:cond-mat/0004380

[56] Jean Schmith, Ney Lemke, José C.M. Mombach, Patrícia Benelli, Cláudia K. Barcellos, and Guilherme B. Bedin. Damage, connectivity and essentiality in protein–protein interaction networks. *Physica A*, 349(3–

4):675–684, April 2005. http://www.inf.unisinos.br/~mombach/protnetv2.pdf

[57] Peter J. Slater. Maximum facility location. *Journal of Research of the National Bureau of Standards B*, 79:107–115, 1975.

[58] Daniel A. Spielman. Spectral graph theory and its applications. Lecture notes, September 2004. http://www-math.mit.edu/~spielman/eigs/

[59] Peter Uetz and Loic Giot. A comprehensive analysis of protein–protein interactions in saccharomyces cerevisiae. *Nature*, 403:623–627, February 2000. http://www.nature.com/nature/journal/v403/n6770/full/403623a0_fs.html

[60] Alexei Vázquez, Romualdo Pastor–Satorras, and Alessandro Vespignani. Internet topology at the router and autonomous system level. *arXiv e–Print*, June 2002. arXiv:cond-mat/0206084

[61] Michele Vendruscolo, Nikolay V. Dokholyan, Emanuele Paci, and Martin Karplus. Small–world view of the amino acids that play a key role in protein folding. *Physical Revue E*, 65(6):061910/1–4, June 2002. http://dokhlab.unc.edu/papers/vdpk_pre02.pdf

[62] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small–world' networks. *Nature*, 393(6684):440–442, June 1998. http://tam.cornell.edu/SS_nature_smallworld.pdf

[63] Stefan Wuchty and Peter F. Stadler. Centers of complex networks. *Journal of Theoretical Biology*, 223(1):45–53, July 2003. http://www.nd.edu/~swuchty/Download/center.pdf

[64] Haiyuan Yu, Dov Greenbaum, Hao X. Lu, Xiaowei Zhu, and Mark Gerstein. Genomic analysis of essentiality within protein networks. *Trends in Genetics*, 20(6):227–231, June 2004. http://papers.gersteinlab.org/e-print/essen/reprint.pdf

[65] Haiyuan Yu, Dov Greenbaum, Hao X. Lu, Xiaowei Zhu, and Mark Gerstein. Supplementary material to [64], February 2004. http://bioinfo.mbb.yale.edu/network/essen/Sup.pdf