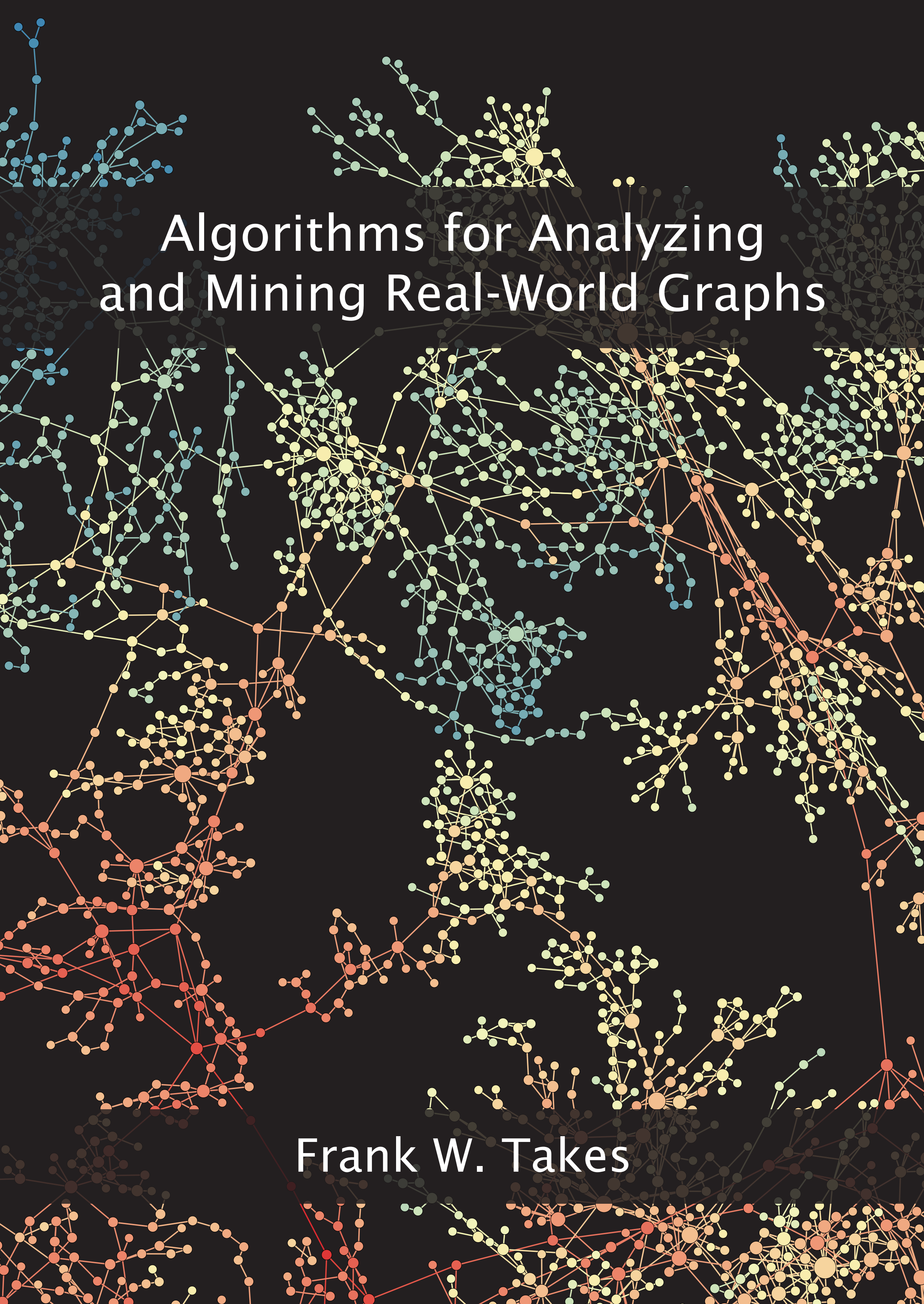

Algorithms for Analyzing and Mining Real-World Graphs

Frank W. Takes

Algorithms for Analyzing and Mining Real-World Graphs

Frank W. Takes

Universiteit
Leiden

The author of this PhD thesis was employed at Leiden University.

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

NWO
Netherlands Organisation for Scientific Research

This research was financed by the Netherlands Organization for Scientific Research (NWO) as part of the Complex Patterns in Streams (COMPASS) project.

Open-access: `https://openaccess.leidenuniv.nl`
Typeset using LaTeX, figures generated using TIKZ and GNUPLOT
Printed by Ridderprint B.V.
ISBN 978-90-5335-957-0

Algorithms for Analyzing and Mining Real-World Graphs

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof.mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op woensdag 19 november 2014
klokke 16.15 uur

door

Frank Willem Takes
geboren te Leidschendam
in 1986

Promotiecommissie

Promotor	prof. dr. J.N. Kok	
Copromotor	dr. W.A. Kosters	
Commissieleden	prof. dr. T.H. Bäck	
	prof. dr. H. Blockeel	(KU Leuven)
	prof. dr. T. Calders	(Vrije Universiteit Brussel)
	dr. H.J. Hoogeboom	
	prof. dr. A.P.J.M. Siebes	(Universiteit Utrecht)

Contents

1 Introduction

1.1 Introduction

We live in a connected world. In our social and digital lives, we are confronted with *networks* (or *graphs*) on a daily basis. When someone tells a story, it is likely that this story passed through various other people that together form a network of social interactions. Online social networks such as Facebook are based on gigantic networks in which people are connected trough so-called friendship links. Browsing the internet means traversing a large network of pages that is connected via clickable (hyper)links. Accessing one webpage on a mobile phone creates a few dozen wired or wireless connections between devices in a matter of microseconds. Networks are everywhere around us, and influence the way in which we communicate, socialize, search, navigate and consume information.

When networks are stored in a digital format, they can produce an enormous amount of *data*. Such a large volume of data is sometimes called *big data*, not only because of its quantity, but also because the data may arrive at an enormous speed and because the data is usually diverse in terms of what type of information it represents. Data is used in many disciplines of science to verify hypotheses about a certain domain. Popularized under the name *data science,* large (network) datasets are being generated and investigated by commercial organizations as well as a number of research disciplines.

Within the field of computer science, we specifically consider tasks related to storing, retrieving, manipulating and understanding data in an automated and efficient way. The most simple type of data is called *unstructured data*, which may for example be the textual content of a news article or numeric measurements from a temperature sensor. On the other hand there is *structured data*, which refers to data that is organized according to some data structure or model (note that other researchers sometimes use the term unstructured data for tabular data, and structured data for graphs). A common example of structured data (according to the first definition) is a *database*, which is made up of tabular structures consisting of different objects (rows) along with attributes (columns) that describe the objects. The majority of this text will however focus on *graphs*, a type of structured data which will be described in Section 1.2. Next, in Section 1.3 the focus will be on algorithms for *computing* certain properties of graph data. Given a dataset, one may also be interested in getting a better understanding of the knowledge incorporated in the data, a task broadly addressed by the field of *data mining*, which will be introduced in Section 1.4. This introductory chapter is concluded in Section 1.5 with an outline of how graphs, algorithms and data mining form the main topics of the following chapters of this thesis.

1.2 Graphs

A *graph* [135] is one of the most fundamental data structures used in computer science. Graphs are used to describe the *relationship* between objects within a certain domain. In a graph, the objects are commonly referred to as *nodes* (also called *vertices*, *actors* or *entities*) and the relationship between two vertices is called an *edge* (also called a *link*, an *arc* or a *tie*). An example of a small graph is shown in Figure 1.1.

This thesis primarily focuses on real-world graphs, often by other disciplines referred to as *networks*. Note that from a computer science perspective, and especially from an algorithmic point of view, the term “graph” is often preferred over “network”, as the latter is often interpreted as a structure of physical connections between multiple devices. A well-known example of a real-world graph is a *social network*, in which a node represents a person, and a link represents a social relationship between the two people that it connects. Throughout this chapter, online social networks are used as a running example to describe the various concepts that are relevant in (real-world) graphs.

Online social networks (*OSN*s) [25] are commonly accessed through a website or (mobile) application, and allow a user to create a profile, and then link this profile to other users, forming a network of social connections called the *friendship graph*. The profile can be enriched with user attributes such as the age, location and gender of the user. Furthermore, the OSN can be used to communicate with other users or to share information by means of for example text, images or video. The first online social networks were introduced around the year 2000, and roughly five years later Facebook, LinkedIn and Twitter were on their way to become online social networking

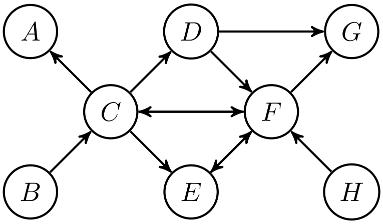

Figure 1.1: A node-labeled unweighted directed graph with 8 nodes and 12 links.

services with over a hundred million members each.

Moving back to the abstract concept of a graph, there are various ways to characterize a graph based on properties of both the nodes and the edges. If a graph contains different types of nodes, it is called *heterogenic*, whereas if all nodes are of the same type, it is called *homogenic*. A homogenic graph is also called a *one-mode network*. A heterogenic graph with two types of nodes is called a *two-mode network* or *affiliation network* if the set of nodes is bipartite, meaning that the node set can be split into two sets of nodes such that for every edge, the source and target node of that edge are in a different set. A two-mode network can be converted into a one-mode network consisting only of nodes of either one of the two types. In the resulting homogenic network, an edge between two nodes (of the one and only type) is present if both nodes linked to the same node of the other type in the original one-mode network. If the relationships in a graph are *explicit*, then this means that both actors explicitly form a connection (as is the case with a link in the friendship graph of for example Facebook). When links are *implicit*, it means that the link is based on some common activity or common property of the two actors, such as the fact that two users sent each other a message. A graph with implicit links is likely to be a projection of a two-mode network.

A node can have one or more *attributes* describing properties of the particular node. In the OSN example this could be the age and location of the person represented by the node. Similarly, an edge can have one or more attributes describing the type of relationship. Graphs with attributes on the nodes or edges are also called *annotated graphs*. If an edge has one numeric attribute, then this edge attribute is often called the *weight* of the edge, and the graph is called a *weighted graph*. In *unweighted* graphs, there is no edge weight (but for computational reasons, the weight of an edge is usually assumed to be equal to one).

In some cases, the direction of a link is relevant, and the graph is called a *directed graph*. This is for example the case in the online social network Twitter, where one user can follow another user, without this other user having to explicitly approve this connection. The term *reciprocity* is used to denote the extent to which links are mutual. Clearly, in a directed graph such as Twitter, reciprocity is only partial. On the other hand, the friendships in Facebook obviously form an *undirected graph*, as two people are always each other's friend, and the relationship is thus always symmetric, realizing full reciprocity by design. The number of incoming links of a node in a directed graph is called the *indegree* (e.g., the number of followers of a Twitter user), and similarly the number of outgoing links is called the *outdegree* of that node (e.g., the number of people a user follows). In an undirected graph, the indegree and outdegree are equal, and there is simply the notion of the *degree* of a node (e.g., the number of friends of a user on Facebook). The majority of this thesis deals with directed or undirected homogenic (one-mode) unweighted graphs.

A common property of the graphs that are investigated in this thesis, is that they are based on real-world data, meaning that the nodes of the graph represent for example actual people, physical objects, locations, organizations, digital information or written articles. An example is the so-called *webgraph*: the "graph of the internet", representing the way in which millions of pages are connected by means of billions of clickable hyperlinks. Other examples are citation and collaboration networks, in which a node represents a scientist, and a link between two nodes indicates respectively a citation (a directed link) or collaboration (an undirected link).

An example of a collaboration network is given in Figure 1.2. In this figure, a node represents a staff member of the computer science department of Leiden University, and an undirected edge between two people indicates that they collaborated between 2005 and 2012 by writing a paper together. In this figure, the edge width is proportional to the number of co-authored papers, and a thinner gray edge indicates indirect collaboration through a common co-author not employed in Leiden. This one-mode collaboration network can be seen as a projection of a two-mode network consisting of authors and publications, with edges connecting publications to their authors.

Furthermore, Figure 1.2 explains the concept of *connected components*: groups of nodes where for any two nodes in this group, there exists a *path* (a sequence of nodes connected through edges) between these two nodes. The figure has three connected components. The *distance* between two nodes is defined as the minimum number of edges that has to be traversed to get from one node to the other, or alternatively, as the length of a *shortest path* between these two nodes. Obviously, this measure of distance has a different semantic meaning depending on the type of graph that is considered. In a collaboration network such as that of Figure 1.2, the distance between two people could be an indication of the similarity of these people's research.

Noteworthy is the fact that many real-world graphs have similar structural properties, even though they are based on completely different data. First of all, real-world graphs are typically *sparse*, meaning that the number of edges is very low compared to the maximum number of edges that may possibly exist between all the nodes. Second,

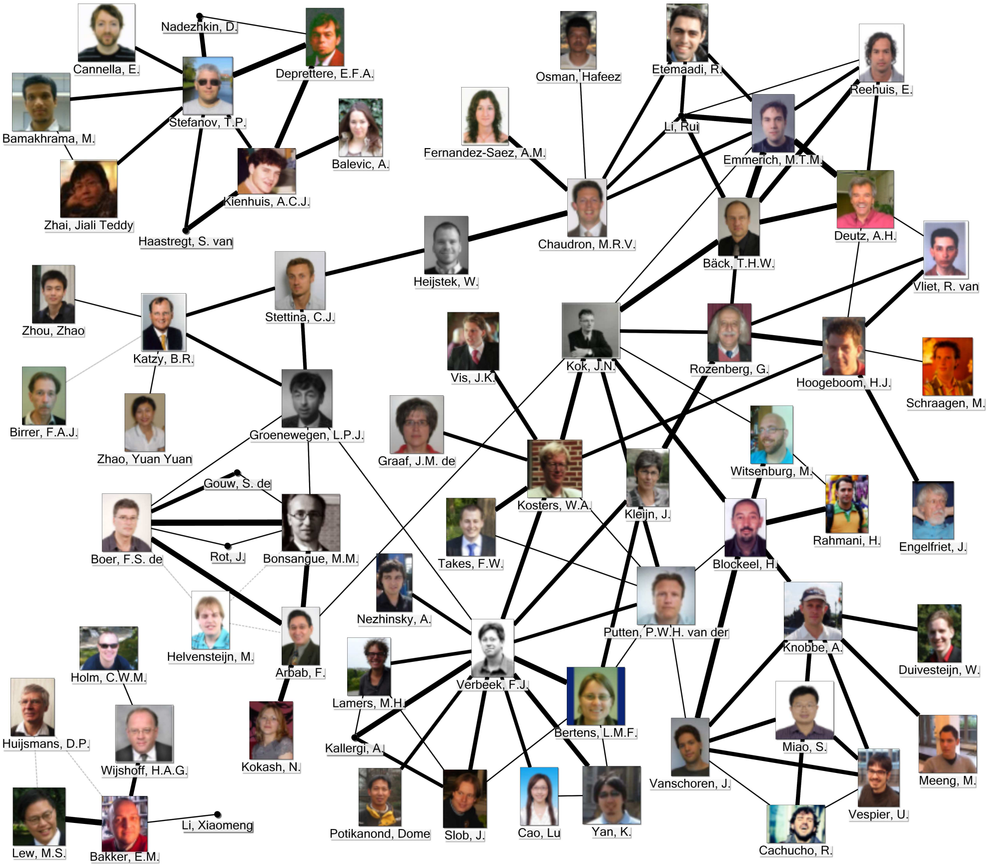

Figure 1.2: A graph of 117 scientific collaborations (undirected edges) between 72 staff members (nodes) of the computer science institute of Leiden University. Data is based on staff publication lists from 2005 to 2012. Visualized using NodeXL (`http://nodexl.codeplex.com`).

these large graphs usually have one large connected component containing the vast majority of the nodes. For example, of a particular online social network considered in Chapter 6, over 99.9% of the in total eight million users is connected via friendship links. Third, most graphs are scale-free, meaning that they have a power-law degree distribution with a fat tail, i.e., there are a lot of nodes with a very low degree, and only a few nodes with a high degree. The fat tail implies that high degree nodes have a degree that is many times larger than the average degree over all nodes. Indeed, high degree nodes of the graph often serve as hubs in the network, realizing very low average node-to-node distances. This fourth commonly observed property is often referred to as the “small-world phenomenon” [71], which is closely related to the concept of “six degrees of separation”, a theory which says that the majority of the people in the world are connected via only six handshakes.

Graphs are studied in many different disciplines of science. Since the sixties, popularized under the name “social network analysis”, networks of social interaction have been extensively studied within the social sciences [139]. There, the goal is to get an understanding of the social interaction between the different actors in a network. Furthermore, physicists refer to large graphs as “complex networks”, and study for example the different models behind networks [7]. It has been shown that the interaction between proteins can be understood by modeling them as a graph [64], demonstrating the applicability of graphs as a model in bioinformatics. Within the field of public administration, large networks of corporations are also studied, for example to model and better understand the global network of corporate control [59]. The structure of such a corporate graph is shown in Figure 1.3, in which a node represents a company in the Netherlands and an edge between two nodes denotes the fact that these companies have a common senior level director.

Indeed, graphs are everywhere, and the interaction between objects that they model is relevant in many areas of research. Whereas other disciplines of science are usually interested in the domain-specific information incorporated in these graphs, for computer scientists, the emphasis is on creating efficient algorithms for storing, analyzing, understanding and computing certain aspects of the graph and addressing the complexity issues that arise when larger graphs are considered.

1.3 Graph algorithms

An *algorithm* [91] can be defined as a sequence of instructions to solve a particular problem. Computer scientists are generally interested in designing algorithms that solve a problem efficiently, both in terms of time and memory usage.

This thesis specifically considers algorithms for *large* graphs. This classification of

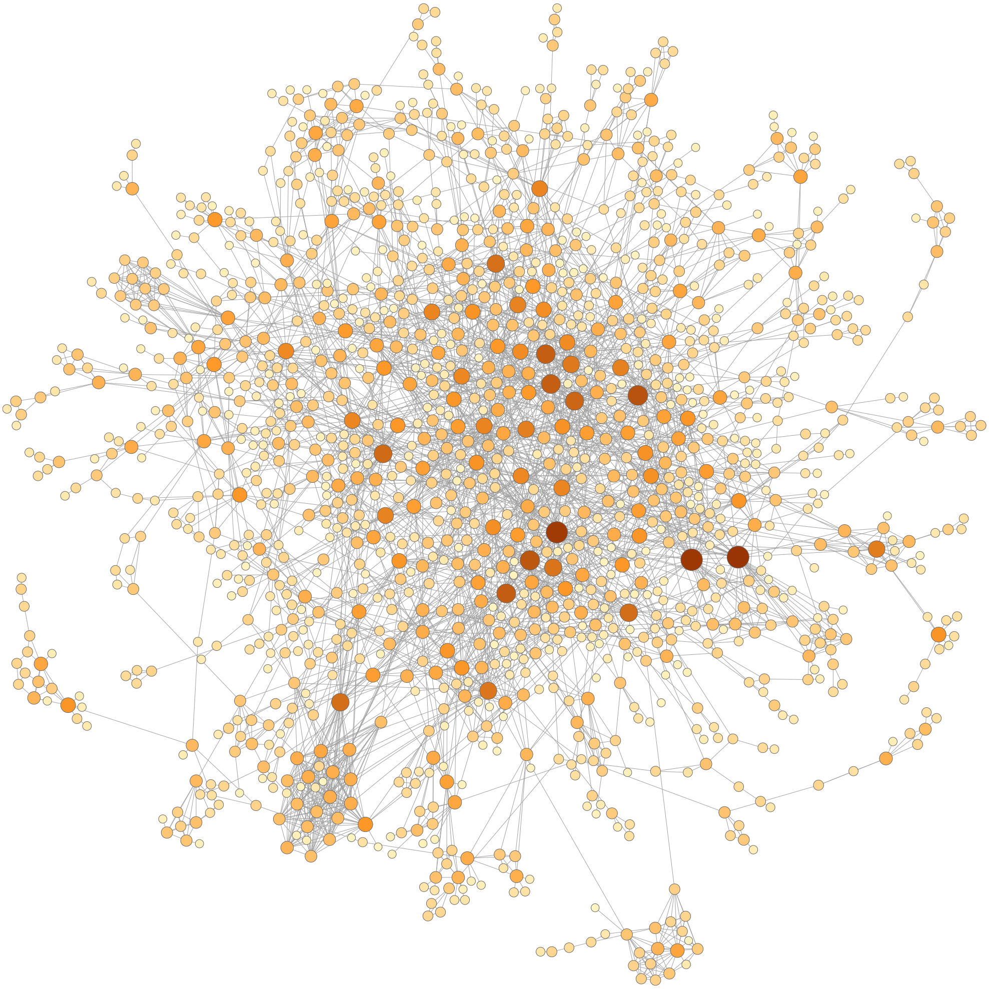

Figure 1.3: A graph of 3,711 board interlocks (undirected edges) between 1,422 companies (nodes) in the Netherlands. Data originates from the ORBIS database of Bureau van Dijk. Visualized using the Fruchterman-Reingold and ForceAtlas2 algorithms in Gephi (`http://gephi.org`).

size may seem rather vague when considering the seemingly ever-increasing amount of available storage, memory and computation power. A more precise way would be to say that large graphs cannot be stored in memory as an adjacency matrix, but only as an adjacency list, making certain operations (such as computing the distance between every pair of nodes) more complex. Algorithms for large graphs usually iterate over the nodes or edges of the graph a constant number of times: quadratic (or worse) complexity in the number of nodes or edges is prohibited, and to ensure practical use the complexity of algorithms should be somewhat linear in the number of nodes or edges. To make these "large" numbers a bit more concrete, large graphs today typically have hundreds of thousands or even millions of nodes, and possibly hundreds of millions or billions of links. Usually, it is very hard to properly visualize graphs once the number of nodes and edges increases. See for example Figure 1.3, which shows a graph consisting of "only" 1,422 nodes and 3,711 undirected edges. Standard tools for visualizing graphs are no longer suitable when the size of the graph exceeds say a hundred thousand nodes. Therefore, when computation or measurements on larger graphs have to be done, specialized frameworks that store and manipulate the graph (without worrying about visualization) are used. For most of the experiments presented in this thesis, a custom C++ framework was used.

A substantial number of graph algorithms proposed in the literature deals with *modeling* or generating graphs using a mathematical model. The focus of graph algorithms discussed in this thesis is on *computing* or measuring certain properties or characteristics of a given (real-world) graph. A well-known example of such a graph algorithm is Dijkstra's shortest path algorithm [43], which computes the distance between two nodes of a graph. Graph properties can roughly be divided into local, global and subglobal properties.

Examples of *global* properties of the graph include the graph diameter (longest shortest path length, see Chapter 2), its average clustering coefficient (the degree to which nodes tend to cluster together on a global scale), or the number of connected components of the graph. On the other hand, *local* properties say something about individual nodes, with a commonly addressed issue being that of *node centrality*, the importance of a node in the graph. In the friendship graph of an online social network, the number of friends of a user (the degree of the node) is a typical centrality measure. The importance of pages in the webgraph of the internet is commonly assessed using the PageRank [107] centrality measure, which ranks webpages based on how many other high-ranked pages link to the considered page. These two measures are incorporated in Figure 1.3, where the size of a node is proportional to its degree (larger size means a higher degree), and the node color corresponds to its PageRank value (darker means a higher PageRank value).

A third type of graph algorithms deals with graphs on a *subglobal* level, computing or deriving aspects of a group of nodes, with *community detection* [89] algorithms as a well-known example. These algorithms try to cluster the nodes in the graph so that groups of nodes that are more connected amongst each other than with the rest of the graph, form one community. Clustering is also a frequently addressed task in the field of data mining, which is the topic of the next section.

1.4 Data mining

Data mining [143] is the field of research that focuses on getting a better understanding of a (large) dataset in an automated way, for example by searching for patterns in the data. The goal is often to find “something new”, i.e., to discover knowledge that is not immediately visible by using common sense or by manually inspecting the data. Alternatively, one could say that data mining deals with converting information into knowledge, a process called *knowledge discovery*. With this in mind, it is often said that data mining is somewhat related to the fields of artificial intelligence, machine learning and statistics. The remainder of this section describes several common data mining tasks, using the small over-simplified database (table) of online social network users from Table 1.1 as an example dataset.

Association is the task of relating attributes of the objects, forming so-called association rules that describe the data. In Table 1.1, a possible association rule could be that if the age attribute has a high value, then the number of friends is low. Indeed, age seems somewhat associated with the number of friends in the example table. *Clustering* refers to the task of grouping sets of objects together because they have certain attributes in common. Again considering the example dataset, a possibility would be to group the users into two clusters based on their gender and location: all female users happen to be from the United States, and all male users are not. *Outlier detection* deals with finding single objects or small groups of objects that “stand out” because they do not comply with the patterns or constraints that the other objects do. A possible outlier in the example dataset could for example be Hugo, because he does

Name	Gender	Age	Location	Photos	Friends
Charlie	male	23	United Kingdom	4	416
Hugo	male	27	Mexico	0	238
Jack	male	42	Australia	8	164
Kate	female	31	United States	815	158
Rose	female	65	United States	39	16

Table 1.1: A small database (table) containing users of an online social network.

not have any photos.

The three techniques described above are all *descriptive*: they attempt to describe or summarize the data, for example to discover the knowledge incorporated in the data, or to find interesting patterns that provide more insight in the considered domain. A more *predictive* task in data mining is that of *classification*: the process of determining, given an object and its attributes, some other (initially unknown) attribute, which is then referred to as the *class* of the object. In the example dataset, the class attribute could be the gender of the user, and one way of predicting this class using the data given in Table 1.1 could be to say that all users with more than 30 photos are female.

For the (too) simple example table, each of the traditional data mining tasks described in this section can be executed more or less perfectly. However, more often than not, a dependency or pattern is only true for a (hopefully large) percentage of the instances, and numeric measures have to be used to assess the accuracy of a derived result. When a set of association rules has been derived, a clustering has been made, or an outlier has been found, it can also be a challenging task to determine whether or not the results make sense within the given context of the data. A clustering might be based on a coincidence in the data, an association rule may be based on a trivial dependency in the attributes, and an outlier may just be an error in the dataset. Therefore in data mining it is important to have a *ground truth* that can be used to verify patterns. Alternatively, one can use separate datasets for training and validating the technique, so that the performance of a certain technique can objectively be measured. Obviously, carefully choosing a correct, suitable and fair ground truth is essential for the verification of results obtained by a data mining algorithm.

When data mining techniques are applied to graph data, we speak of *link mining* [49] or *graph mining* [33]. A well-known predictive task in this context is that of *link prediction*: given a graph of existing nodes and edges, which edges are likely to appear (or disappear) in the future? A common descriptive graph mining task is *frequent subgraph* detection: given a graph, which subgraph occurs more frequently than expected, and may indicate a pattern in the graph? For each of these tasks, it is important to keep the network aspect of the data in mind: it is not only the objects and their attributes, but also the relationships between the objects that may define the knowledge that is incorporated in the data.

1.5 Thesis outline

This thesis consists of two parts. Part I deals with efficient computation of distance-related measures and properties of graphs. The algorithms and techniques introduced in this first part help answer questions such as:

- What is the diameter of a given real-world graph? (Chapter 2)
- How can we determine which nodes form the center of a large graph? (Chapter 3 and Chapter 4)
- How can we efficiently assess the distance between two pages on the internet? (Chapter 5)
- What measures and techniques are able to identify the prominent actors in an online social network? (Chapter 6)

In Chapter 2, an algorithm for efficiently computing the exact diameter of large graphs is introduced, and a similar technique is used in Chapter 3 to also compute the eccentricity distribution. Chapter 4 provides a generalized version of the algorithms presented in the previous two chapters to efficiently compute various other distance measures including the radius, center and periphery of a graph. In each of these chapters, shortest paths in graphs are exactly computed. To speed up this process at the cost of exactness, in Chapter 5 so-called landmark strategies are discussed, which can be used to approximate the distance between two nodes with high accuracy. Finally, Chapter 6 can be seen as a case study in which the (former) Dutch online social network Hyves is considered in the context of so-called centrality measures that determine the importance of a node in a graph.

Part II of this thesis is more oriented towards data mining in information networks, as both chapters are based on data from users that are navigating the well-known free online encyclopedia Wikipedia. The second part addresses issues related to the following questions:

- How difficult is it for humans to navigate through a network of Wikipedia articles? (Chapter 7)
- What are the differences between human search strategies and algorithmic search strategies? (Chapter 7)
- What can be learned from the patterns in human navigation paths in information networks? (Chapter 8)

In Chapter 7 and Chapter 8, the studied dataset of human traversal patterns originates from the Wiki Game, an online game in which the main task is to link two given random Wikipedia articles by means of clicking the hyperlinks between these pages.

Chapter 7 focuses on both failed and successful user-generated paths in order to assess the difficulty of this path finding task, whereas Chapter 8 considers mining the successful paths in an attempt to better understand the human search strategy.

Each of the seven chapters following this introduction ends with a conclusion, summarizing the results presented in that chapter and providing suggestions for future work.

Publications

The different chapters of this thesis are based on the following peer-reviewed publications:

- F. W. Takes and W. A. Kosters. Determining the diameter of small world networks. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM 2011)*, pages 1191–1196, 2011 (Chapter 2)
- F. W. Takes and W. A. Kosters. Computing the eccentricity distribution of large graphs. *Algorithms*, 6(1):100–118, 2013 (Chapter 3 and Chapter 4)
- F. W. Takes and W. A. Kosters. Adaptive landmark selection strategies for fast shortest path computation in large real-world graphs. In *Proceedings of the IEEE/ACM International Conference on Web Intelligence (WI 2014)*, pages 27–34, 2014 (Chapter 5)
- F. W. Takes and W. A. Kosters. Identifying prominent actors in online social networks using biased random walks. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)*, pages 215–222, 2011 (Chapter 6)
- F. W. Takes and W. A. Kosters. The difficulty of path traversal in information networks. In *Proceedings of the International Conference on Knowledge Discovery and Information Retrieval (KDIR 2012)*, pages 138–144, 2012 (Chapter 7)
- F. W. Takes and W. A. Kosters. Mining user-generated path traversal patterns in an information network. In *Proceedings of the IEEE/ACM International Conference on Web Intelligence (WI 2013)*, pages 284–289, 2013 (Chapter 8)

A full list of publications by the author can be found on page 167 of this thesis.

Part I

Graph Algorithms

2

Determining the Diameter of Small-World Networks

This chapter presents a novel approach to determine the exact diameter (longest shortest path length) of large graphs, in particular of the nowadays frequently studied small-world networks. Typical examples include social networks, biological networks, webgraphs and internet topology networks. Due to complexity issues, the diameter is often computed based on a sample of only a fraction of the nodes in the graph, or some approximation algorithm is applied. Instead, we propose an exact algorithm that uses various lower and upper bounds as well as effective node selection and pruning strategies in order to evaluate only the critical nodes which ultimately determine the diameter. The proposed algorithm is able to quickly determine the exact diameter of various large small-world networks with millions of nodes and hundreds of millions of links, whereas before only approximations could be given. This chapter is based on:

- F. W. Takes and W. A. Kosters. Determining the diameter of small world networks. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM 2011)*, pages 1191–1196, 2011

2.1 Introduction

With the rapidly increasing amount of graph data that is being generated and academically studied, researchers are often interested in quickly deriving various global properties of their graphs. While several trivial static properties of the graph such as the graph density (number of edges of the graph vs. the maximum number of edges that could possibly exist) can easily be computed, determining other properties of large graphs using straightforward algorithms may require a lot more computation time. One of these more "expensive" properties is the *diameter* of a graph, which is defined as the maximal distance (length of a longest shortest path) between any two nodes in the graph. Exact algorithms for computing the diameter traditionally require running an All Pairs Shortest Path (APSP) algorithm for each node in the graph, ultimately returning the length of one of the longest shortest paths that was found. While this will indeed return the exact diameter of the graph, complexity for a graph with n vertices and m edges is in the order $O(n^3)$ for weighted graphs and $O(mn)$ for sparse unweighted graphs. This naive method for obtaining the diameter is clearly not feasible in extremely large graphs with for example millions of vertices and a billion edges.

We will study the diameter of *small-world networks*: sparse networks that are most typically characterized by an average distance between two random nodes that grows only proportionally to the logarithm of the total number of nodes in the network [71]. Examples of small-world networks that are frequently studied are webgraphs [8, 28], internet topology networks [66] and biological networks [6, 67], but perhaps nowadays most well-known are social networks [120, 139]. With the introduction of *online* social networks such as Facebook, LinkedIn and Orkut, even more than before, the study of social networks has become interesting for computer scientists, as the data behind these networks can easily be gathered in a digital format. Other (implicit) social networks are telephone call graphs, e-mail networks [73] and scientific collaboration networks [13].

The diameter is a relevant property of a network for many reasons. For example in social networks, the diameter could be an indication of how quickly information reaches literally everyone in the network. Within a scientific collaboration network, a high diameter may indicate that there are groups of researchers that are not working together very closely. In an internet routing network, the diameter could reveal something about the worst-case response time between any two machines in the network. In a way, the diameter can be seen as a measure of how data or information spreads over the network in the worst case.

The diameter is not just a static property of a graph, but it is also used in various algorithms in which it serves as the maximum depth of a search procedure, for example in a Depth Limited Search algorithm [117]. Work is also being done on studying how a graph evolves over time [42]. There, knowing the exact point in time when the diameter changes can be interesting, which may favor an exact answer over an approximation. Another important advantage of studying the exact diameter is that we can observe the actual path that realizes the diameter, a piece of information that we do not get when for example an approximation algorithm is used, or when the diameter is estimated by looking at a sample.

The main contribution of this chapter consists of a new algorithm for determining the exact diameter of small-world networks. Based on various lower and upper bounds and critical node selection strategies, we improve upon the straightforward APSP algorithm as well as upon existing approximation algorithms, obtaining the exact diameter of networks with millions of nodes in a matter of seconds or minutes. The performance is empirically verified on various large small-world networks.

The rest of the chapter is structured as follows. Section 2.2 introduces various definitions and a short analysis of the problem's complexity, after which we will cover relevant related work in Section 2.3. We will use Section 2.4 to outline our algorithm for deriving the diameter of a graph, and discuss experimental results in Section 2.5. In Section 2.6 we look at a parallel version of the proposed algorithm, and finally Section 2.7 concludes.

2.2 Preliminaries

In this section we will first consider some basic definitions related to graphs, distances, eccentricity, graph diameter and shortest path problems, and then give some insight in the complexity of determining these measures. After that we will briefly discuss small-world networks.

2.2.1 Definitions

A graph $G = (V, E)$ consists of a set of vertices (or *nodes*) V and a set of links (or *edges*) $E \subseteq V \times V$. Throughout the chapter we will use n to denote the number of nodes $|V|$, and for the number of links $|E|$ we use m. The distance $d(v, w)$ between two nodes $v, w \in V$ is defined as the length of a shortest path from v to w. This chapter deals with *unweighted* graphs, and we will assume that graphs are *undirected*, meaning that $(v, w) \in E$ iff $(w, v) \in E$ and thus $d(v, w) = d(w, v)$. Our definition of an edge does not allow parallel edges, and we furthermore disallow self-loops. Thus note that m is the number of (directed) links between distinct nodes and $m/2$ is the number of (undirected) edges. The *degree* of a node v in an undirected graph is simply defined as the number of (undirected) edges connected to that node. Finally, we assume that the graph is *connected*, implying that for each $v, w \in V$, $d(v, w)$ is a finite number. We are now ready to define the two most important concepts used in this chapter, node *eccentricity* and graph *diameter*:

Eccentricity The *eccentricity* $e(v)$ of a node $v \in V$ is defined as $\max_{w \in V} d(v, w)$: the length of a longest shortest path starting at node v.

Diameter The *diameter* $D(G)$ of a graph G is defined as $\max_{v,w \in V} d(v, w)$: the longest shortest path length between any pair of nodes, or equivalently as the maximum eccentricity over all nodes: $\max_{v \in V} e(v)$.

Note that when used as a variable, we simply use D to denote the diameter. For convenience, we define two combinatorial problems that are frequently addressed in this chapter and are tightly related to eccentricity. First, the *Single-source Shortest Path* (*SSP*) problem is the problem that deals with finding all shortest paths from a single source node $v \in V$ to all other nodes in the graph. For non-sparse graphs this problem has the traditional time complexity of $O(n^2)$ (Dijkstra's shortest path algorithm). When the graph is sparse, it can more efficiently be stored using an adjacency list instead of an adjacency matrix. Then in our unweighted case, time complexity can even be reduced to $O(m)$, as a Breadth First Search (BFS) from the starting node is sufficient to find all shortest paths starting at that node. In essence, solving the SSP problem for a node means that we have found the eccentricity of that particular node.

Next, we can define the *All Pairs Shortest Paths* (*APSP*) problem as the problem of finding the shortest paths between all pairs of nodes of the graph, which increases the previous time complexity by a factor n to $O(n^3)$ for weighted graphs, and $O(mn)$ for the considered sparse unweighted graphs. The maximum distance value that the APSP algorithm obtains, is then the maximum eccentricity (which is computed for a node v using the function ECCENTRICITY() in Algorithm 2.1) over all nodes and thus equal to the diameter of the graph. So if we solve the APSP problem, we have also found the diameter of the graph.

Algorithm 2.1 DIAMETERAPSP

```
Input: Graph $G$
Output: Diameter of $G$

$D \leftarrow -\infty$
for $v \in V$ do
    $D \leftarrow \max(D, \text{ECCENTRICITY}(v))$        // one BFS
end for

return $D$
```

2.2.2 Small-world networks

In this chapter we will specifically look at the diameter of *small-world networks*. A good overview of algorithmic properties of these networks, of which we will discuss a few, is given in [71]. First of all, small-world networks are generally *sparse*: the total number of links m is very small compared to the maximum number of links $n(n-1)$. This may cause the reader to believe that nodes have a rather long shortest paths between them, as there are very few links in general. However, a second interesting characteristic is that even though the network is very sparse, the average distance between two nodes is very small. More specifically, this distance is typically somewhat proportional to the logarithm of the total number of nodes. The node degree distribution of a small-world network usually follows a power law: there are only a few nodes with a very large number of connections, the so-called *hubs*, and there are many nodes with relatively few connections. Hubs are in turn responsible for realizing very low average shortest path lengths. So even though many nodes are not direct neighbors of one another, most nodes can be reached from every other node via only a small number of steps. A last property of small-world networks, is that they generally contain one very large connected component (the *giant component*) which contains the vast majority of the nodes.

2.3 Related work

A lot of work has been done on devising algorithms for the *estimation* of the diameter [44, 115]. Such estimation algorithms typically determine the diameter of any type of graph (sparse or dense) with some very small additive error, but using significantly less computation time than the APSP algorithm. For example in [3], a method is suggested which finds the diameter in $O(n^{2.5}\sqrt{\log n})$ time with an additive error of 2. Work has also been been done on testing if the diameter is (with some small margin of error) equal to a certain value [110].

A popular method which is used in many graph analysis toolkits, is the Approximate Neighborhood Function (ANF) by Palmer et al. [109]. This technique approximates the size of the neighborhood of (sets of) nodes, and is thus also able to approximate the diameter. Based on this technique, a variant of the diameter called the *effective diameter* was introduced, which is defined as the 90-th percentile of the cumulative distribution of shortest path lengths. Though this measure may appear more robust to outliers, it is claimed that the diameter and the effective diameter “tend to exhibit qualitatively similar behavior” [86]. Another measure closely related to the diameter that is sometimes mistakenly spoken of as if it were the real diameter, is what some call the average diameter, which is actually the average shortest path length: the average distance between any pair of nodes, i.e., $1/(n(n-1))\sum_{v \neq w \in V} d(v, w)$. This value is often approximated by selecting a few thousand random pairs of nodes from the graph, and determining the average of their pairwise distances.

In work which does not focus on actually determining the diameter, but where it is found only as a static property of the dataset, a *sample* of the graph is frequently used to determine the diameter [84, 103]. There are at least two directions to determining the diameter when using a sample. The first option would be to select a sample of the nodes in the original network using a suitable sampling approach [84], and then determining the diameter of this sample using the straightforward APSP algorithm. The second option would be to assess the diameter based on selecting a few nodes from the original graph that are likely to have a high eccentricity value, which is somewhat the idea behind the method described in [88]. In this work, the diameter is determined by, starting from a random node, repeatedly selecting the farthest node, meanwhile keeping track of the highest distance so far. If this value no longer increases after a certain number of iterations, then this value is a lower bound on the diameter has been found. Similar techniques are employed in [17, 34, 95], and it is argued that using a handful of Breadth First Searches, empirically tight bounds on the diameter can be obtained.

Most *exact* algorithms for finding the diameter are actually implementations of matrix multiplication that solve the APSP problem and thus also find the diameter. While these algorithms work well and have time complexity $O(n^{2.376})$ [9], they usually suffer from large hidden constants, and are often very unpractical due to large memory requirements. To the best of our knowledge, the exact approach suggested by Crescenzi et al. [36, 37] is the only other exact algorithm for determining the diameter of large graphs. The suggested approach uses a strategy somewhat similar to the algorithm that we propose in this chapter. A comparison is provided in [36, 99].

2.4 BoundingDiameters

In this section we describe our approach for computing the diameter. We will start with some observations about the eccentricity of neighboring nodes and how they influence the diameter. Then we describe the actual algorithm called BOUNDINGDIAMETERS, which makes use of these observations to improve upon the APSP algorithm. Next we discuss the algorithm's complexity and some simple optimization techniques.

2.4.1 Observations

If we compute the eccentricity $e(v)$ for some node v, we know that for all nodes w with $d(v, w) = k$, their eccentricity $e(w)$ lies between $e(v) - k$ and $e(v) + k$. The upper bound follows because any node w at distance k of v can get to v in exactly k steps, and then reach any other node in at most $e(v)$ steps. The lower bound can be derived in the same way, by interchanging v and w in the previous statement. In the "best" case, w is on some path that realizes the eccentricity of v, and has an eccentricity $e(w)$ of only $e(v) - k$. This lower bound can of course never be less than k itself: if the shortest path between v and w has length k, then $e(w)$ is at least equal to k. In essence, we are making use of the triangle inequality in graphs. So we have:

Observation 2.1 ***Node eccentricity bounds***
If a node $v \in V$ has eccentricity $e(v)$, then for all nodes $w \in V$ we have:

$$\max(e(v) - d(v,w), d(v,w)) \leq e(w) \leq e(v) + d(v,w)$$

The proof of this observation is simple: we know that the diameter of a graph is equal to the maximum eccentricity over all nodes. Therefore, the maximum lower bound on the eccentricity over all nodes, is also a lower bound for the diameter. Similarly, the maximum upper bound on the eccentricity over all nodes can be seen as an upper bound on the diameter. The upper bound can even be made more tight by observing that this bound can be at most twice as big as the smallest eccentricity upper bound over all nodes, as also observed in [95]. These observations can be formalized as follows to form lower and upper bounds on the diameter:

Observation 2.2 ***Diameter bounds***
Let $e_\ell(v)$ and $e_u(v)$ denote currently known lower and upper bounds for the eccentricity of node $v \in V$. For the diameter $D(G)$ of a graph G it holds that:

$$\max_{v \in V} e_\ell(v) \leq D(G) \leq \min(\max_{v \in V} e_u(v), 2 \cdot \min_{v \in V} e_u(v))$$

We will denote these lower and upper bounds on the diameter by D_ℓ and D_u, respectively. Note that as opposed to the even upper bound of $e(v) \leq D(G) \leq 2 \cdot e(v)$ suggested in [95], the proposed algorithm is able to derive an odd upper bound, which is obviously necessary for finding the exact diameter when the diameter itself has an odd value.

2.4.2 Algorithm

The bounds mentioned above can be used to improve the original APSP algorithm from Algorithm 2.1 by reducing the number of eccentricity computations, as only nodes that can actually contribute to the diameter bounds are considered. Pseudocode for the BoundingDiameters algorithm is given in Algorithm 2.2.

After setting some initial values (lines 3–7), in the main while-loop, the algorithm repeatedly selects a node v (line 9) from the candidate set W, which initially contains all the nodes. The various mechanisms for selecting the next node to be examined are outlined in Section 2.4.4. The algorithm then computes the eccentricity of that node v (line 10), and uses the result to update the lower and upper bound of the graph diameter (D_ℓ and D_u, lines 11–12), cf. Observation 2.2. Note that we use $e[v]$ when we reference to the (array) variable containing the eccentricity $e(v)$ of node v. Next, the eccentricity bounds of all nodes in the candidate set W are updated (lines 14–15) according to Observation 2.1. Then we determine which nodes (including v) can be removed from the set of candidates (line 17). This can happen either because the eccentricity of the node is already known since the lower and upper bounds are identical (which is always the case for the current node v), or because a node can no longer "contribute" to the diameter of the graph by increasing the lower bound or decreasing the upper bound (line 16). Note that because we performed a BFS to compute the eccentricity of v, we know for each node w the distance $d(v, w)$ which is needed to apply Observation 2.1. After adjusting the node eccentricity bounds, the diameter upper bound is further tightened using the largest node eccentricity upper bound, again cf. Observation 2.2 (line 20). Finally, the algorithm stops when all nodes have been examined, or when the lower bound is equal to the upper bound (line 8). It then returns the lower bound of the diameter, which at that point contains the real value of the diameter (line 22).

A proof of the correctness of this algorithm can be constructed by considering the fact that D_ℓ, which is returned on line 22, contains the largest computed eccentricity value (line 12). So, given Observation 2.1 and the assumption that in line 16 only nodes that can not potentially increase the value of D_ℓ are removed, the algorithm returns the correct value of the diameter.

2.4.3 Complexity

Computing the eccentricity of a node using the ECCENTRICITY() function (line 10) is the critical operation of the algorithm, as this requires running a SSP algorithm (one BFS), taking $O(m)$ time. In the best case, we only have to compute the eccentricity of two nodes v and w, only to find that $e(w) = 2 \cdot e(v)$ (or vice versa), which means that the diameter is equal to $e(w)$. In the worst case the algorithm needs to investigate the eccentricity of every single node, not improving the traditional APSP time complexity of $O(mn)$. An example of a graph in which all nodes have to be investigated in order to determine the diameter, is a graph where the nodes are connected through exactly one circle of edges, meaning that all nodes have identical eccentricity. Of course, graphs are generally not shaped as a circle, neither is the diameter always equal to two times

Algorithm 2.2 BoundingDiameters

```
Input: Graph $G$
Output: Diameter of $G$

$W \leftarrow V \quad D_\ell \leftarrow -\infty \quad D_u \leftarrow +\infty$
for $w \in W$ do
    $e_\ell[w] \leftarrow -\infty$
    $e_u[w] \leftarrow +\infty$
end for

while $D_\ell \neq D_u$ and $W \neq \emptyset$ do
    $v \leftarrow$ SelectFrom($W$)
    $e[v] \leftarrow$ Eccentricity($v$)

    $D_\ell \leftarrow \max(D_\ell, e[v])$
    $D_u \leftarrow \min(D_u, 2 \cdot e[v])$

    for $w \in W$ do
        $e_\ell[w] = \max(e_\ell[w], \max(e[v] - d(v,w), d(v,w)))$
        $e_u[w] = \min(e_u[w], e[v] + d(v,w))$
        if $(e_u[w] \leq D_\ell$ and $e_\ell[w] \geq D_u/2)$ or
            $(e_\ell[w] = e_u[w])$ then
            $W \leftarrow W - \{w\}$
        end if
    end for

    $D_u \leftarrow \min\left(D_u, \max_{w \in V}(e_u[w])\right)$
end while

return $D_\ell$;
```

the eccentricity of some node in the graph (if we are even able to quickly find two such nodes). In general, the eccentricity values of nodes in a network differ, and how much they differ will likely influence the number of iterations that is required, as larger differences in eccentricity values will result in tighter eccentricity bounds on surrounding nodes.

We claim that the algorithm specifically works well on small-world networks, which we believe is due to power law degree distribution within such networks, as discussed in Section 2.2.2. A small-world network has relatively few nodes with a very high degree (hubs), that will often (but not always) have a relatively low eccentricity value. The remainder of the nodes typically have a much lower degree, often (again, not always) resulting in a relatively high eccentricity value. Thus, due to the expected existence of a large diversity in eccentricity values of the nodes in a small-world network, the bounds on the diameter will typically converge very quickly. When we look at a degree-based selection strategy in Section 2.4.4.1, we will verify this claim empirically.

2.4.4 Selection strategies

This section describes the different strategies that can be used to select the next node for which we want to compute the eccentricity, outlining the possible functionality of the SELECTFROM() function that is called in line 9 of Algorithm 2.2. First notice how enumerating the nodes in some order, or selecting them at random, will in essence mean that we are executing the APSP algorithm, only now we discard nodes that can no longer contribute to the diameter bounds. While this will no doubt already improve upon the APSP algorithm, it mainly serves as a baseline of comparison, as the main objective is to tighten the bounds of the diameter as quickly as possible in order to efficiently reduce the size of the set of candidate nodes.

Any strategy that we come up with has to be easy to compute so that it does not influence the overall complexity of the diameter algorithm. More specifically, it should be possible to determine the next node by iterating over the set of nodes once, such that the selection function could even be done on-the-fly while updating the bounds in the previous iteration. We will test the performance of (combinations of) the strategies described below in Section 2.5.

2.4.4.1 Degree centrality

Perhaps the simplest strategy would be to select nodes based on their *degree*, hoping that high degree nodes have a low eccentricity value, and vice versa. This measure, known as *degree centrality*, is often suggested as a simple measure of the centrality of a node within a network, but is far from perfect for predicting the eccentricity. For example, in Figure 2.2 node F has the highest degree (6 links) and eccentricity $e(F) = 5$, whereas node J with lower degree 3 has eccentricity $e(J) = 4$. Also, a low degree is no guarantee for a low eccentricity value, as in small-world networks there are typically many nodes with a low degree, and these nodes may still be connected to the most central nodes, resulting in a low eccentricity value even though the degree is also very low. The problem here is that degree centrality is merely a local measure: it does not take into account any aspects of the graph beyond its own neighborhood. Indeed, as Figure 2.1 suggests, node degree and node eccentricity are not directly related: not all nodes with a high degree have a low eccentricity value, and not all nodes with a low degree have a high eccentricity value. Therefore we suggest using the degree as a secondary selection mechanism only, mainly to break ties in other selection methods, or to select the very first node to be examined. We mention that although many more centrality measures exist [27], a downside is that they are often as hard to compute as the eccentricity or the diameter itself and therefore not suitable to serve as a selection mechanism.

2.4.4.2 Eccentricity bound difference

During the execution of the proposed algorithm, the *difference* $e_u(v) - e_\ell(v)$ between the lower and upper eccentricity bound could be an interesting feature, as it says something about how much we already know about the eccentricity of node v and its neighborhood. If for a certain node this difference is very big, determining its eccentricity may tighten the bounds of many nearby nodes. In essence, sorting by bound difference in decreasing order means that we are repeatedly taking a node in an area of the graph which has not been very thoroughly explored yet.

2.4.4.3 Interchanging eccentricity bounds

Inspired by traditional branch-and-bound algorithms that repeatedly select the nodes with the best bound value for expansion, we could choose to select nodes from the candidate set based on their their quality in terms of how well we expect them to

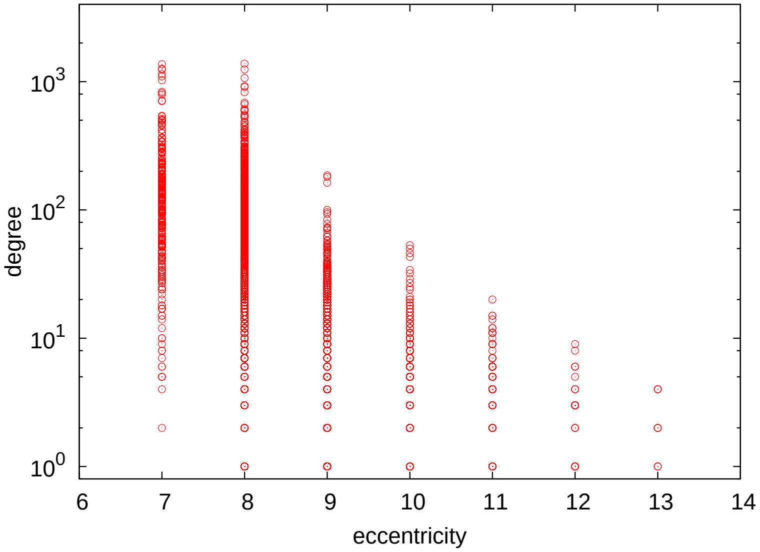

Figure 2.1: Degree (vertical axis) and eccentricity (horizontal axis) of the nodes in the ENRON graph.

contribute to tightening the diameter bounds. To find nodes with high eccentricity, we can select the node v with the largest upper bound $e_u(v)$, and similarly we choose a node with a small lower bound $e_\ell(v)$ to find nodes with a low eccentricity value. As the goal is to increase the lower bound and decrease the upper bound, we propose to *interchange* the selection of the node with the smallest lower bound and the node with the largest upper bound.

2.4.4.4 Repeated farthest distance

Another option is to select a node based on its distance to the previously investigated node, and then select a node with the highest distance. So starting from some initial node v, we repeatedly select the farthest possible candidate node w with $d(v, w) = e(v)$. This is a variation of the heuristic for approximating the diameter suggested in [88]. The only difference is that in this context, the stopping criterion for the algorithm is exactly defined, namely when the diameter lower and upper bounds are equal.

2.4.5 Example

We will give an example of how BOUNDINGDIAMETERS would determine the diameter of the graph depicted in Figure 2.2. As a selection strategy we alternately choose the largest upper bound and smallest lower bound (cf. Section 2.4.4.3), breaking ties by choosing the nodes with the highest degree (cf. Section 2.4.4.1). Any remaining ties are broken by choosing a random node. In this example, we will denote the lower and upper eccentricity bounds $e_\ell(v)$ and $e_u(v)$ of a node v by $[e_\ell(v); e_u(v)]$.

Initially, all nodes form the candidate set, and all lower bounds and all upper bounds are equal. We start at node F which has the highest degree, and remove it from the candidate set. The situation of Figure 2.2 depicts the situation after the first iteration, where node F has been investigated. The diameter lower and upper bounds are now equal to $D_\ell = e(F) = 5$ and $D_u = 2 \cdot e(F) = 10$, respectively. Notice how for node M and N we set the bounds to $[3; 8]$ and not to $[2; 8]$, because $d(M, F) = d(N, F) = 3$ and the eccentricity is at least equal to 3. The current eccentricity bounds do not yet require us to remove any nodes from the candidate set.

In the second iteration, we determine the eccentricity of the node with the largest eccentricity upper bound, which could be T or S as they both have bounds $[5; 10]$. We choose T. The eccentricity of node T turns out to be 7, and the eccentricity bounds after the second iteration are depicted in Figure 2.3. Here, nodes that can no longer contribute to computing the diameter are green if the lower and upper bounds have become equal and red if they have bounds such that they cannot contribute to either increasing the lower bound or decreasing the upper bound. The graph diameter now

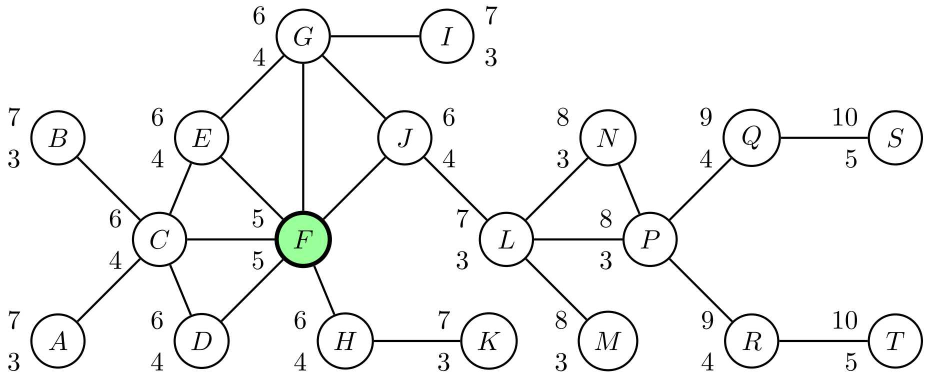

Figure 2.2: Example graph with eccentricity bound values after the first BFS from F.

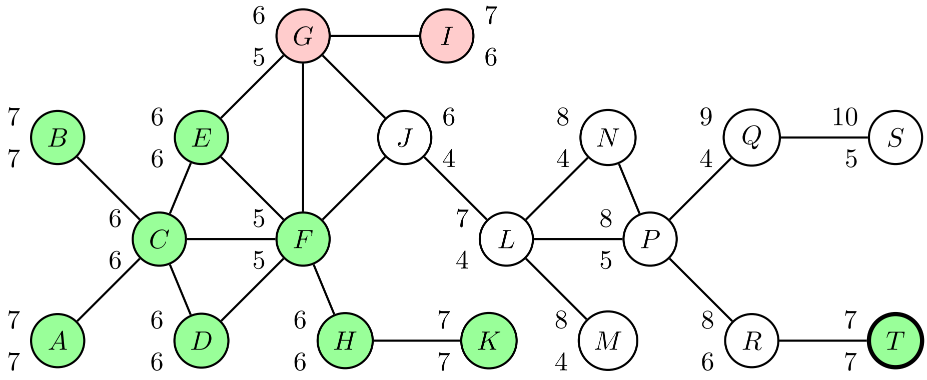

Figure 2.3: Example graph with eccentricity bounds after the second BFS from T.

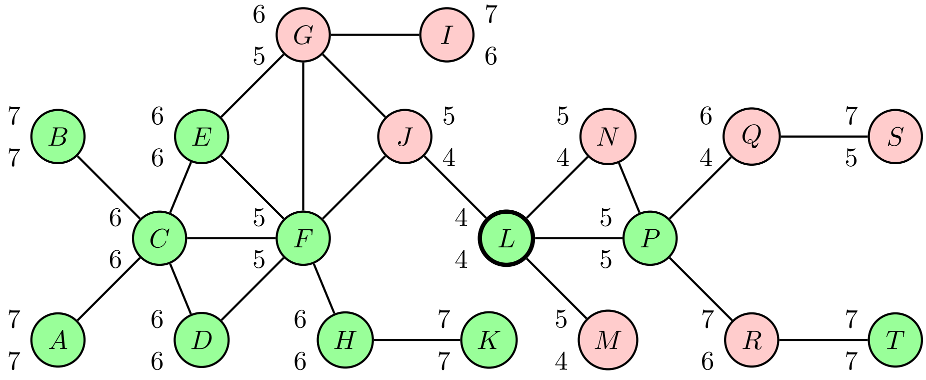

Figure 2.4: Example graph with eccentricity bound values after the third BFS from L.

lies between $D_\ell = 7$ and $D_u = 10$. We can now remove A, B, C, D, E, H and K from the candidate list, as we have found the exact eccentricity of these nodes (but without having computed it explicitly). We can also remove node I and node G with bounds $[6; 7]$ because they can no longer contribute to raising the lower bound or decreasing the upper bound.

For the third loop of the algorithm we compute the eccentricity of the node with the smallest lower bound (and as secondary selection, the highest degree), which is node L. It has an eccentricity of 4, meaning that we can now discard all nodes based on the same arguments as in the previous iteration, resulting in all nodes being visited (see Figure 2.4), terminating the algorithm after only 3 eccentricity computations, and returning the maximum over all lower bounds as the final value of the diameter: 7.

2.4.6 Pruning

The size of the graph can be reduced by applying the following pruning strategy beforehand. For every node we can determine if removing all of its adjacent edges would disconnect the graph. If this is the case, and multiple identically structured small subgraphs remain, we can remove each but one of them, and still obtain the correct diameter value, assuming of course that a path that realizes the diameter of the graph does not run from one subgraph to another (pruned) subgraph. Therefore, the diameter of the pruned subgraph has to be smaller than $D(G)/2$.

For example node C in Figure 2.2 is connected to two identical subgraphs, namely the subgraph consisting of node A and the subgraph consisting of node B. We could prune one of these subgraphs, as they will both have identical eccentricity (bound) values. Similarly, P is connected to identical subgraphs $Q - S$ and $R - T$, both with identical eccentricity values. Indeed, in the second iteration of the example run described in Section 2.4.5, we could have chosen either S or T, both resulting in the same adjustments to the bound values of the remaining nodes.

The proof of the validity of this pruning strategy can be constructed based on the concept of graph isomorphism, a bijection from one graph to another graph in which the connectedness of the graph is preserved. More precisely, a *graph isomorphism* $h : G \to G'$ of a graph $G = (V, E)$ to another graph $G' = (V', E')$ is a bijection $h : V \to V'$ from the set of nodes V in the original graph to the set of nodes in the projected graph V' such that $(u, v) \in E$ iff $(h(u), h(v)) \in E'$ [60]. In the example from the previous paragraph, node A and B map to each other, as do nodes Q and S to nodes R and T, respectively. Because graph isomorphism preserves connectedness, distance measures such as the eccentricity are also preserved. Although the general problem of deciding if there exists a isomorphism from one graph to another graph is NP-complete, the strategy described above is able to efficiently detect the simple type of isomorphism, namely that of subgraphs of a very small size that arise when only one edge is removed.

Nodes that are pruned contribute to speeding up the algorithm in two ways: pruned nodes do not have to be considered during the eccentricity computation and also do not have to be included in the set of candidate nodes.

2.5 Experiments

This section starts with a brief description of the datasets (graphs), and then describes a measurement methodology for the different selection strategies described in Section 2.4.4. Next, the results of applying these strategies in the BoundingDiameters algorithm are discussed.

2.5.1 Datasets

We will verify the algorithm on various small-world networks. Characteristics of these graphs, such as the number of nodes, the number of links, the average degree $\overline{deg}$, average node-to-node distance $\overline{d}$ and the diameter $D(G)$, are given in Table 2.1. Numbers are based solely on the largest connected component of each graph, and originally directed graphs are interpreted as if they are undirected. Therefore, slight deviations from statistics presented in the original papers describing these graphs may be observed.

The CA-AstroPh dataset is a an undirected network of scientific collaborations (co-authorship) in the field of astrophysics, which was obtained through arXiv and analyzed in [87]. Enron [73] is a well-known network of e-mail contacts within a

Dataset	**Nodes** n	**Links** m	$\overline{deg}$	$\overline{d}$	$D(G)$
CA-AstroPh [87]	17,903	393,944	21	4.15	14
Enron [73]	33,696	361,622	10	4.07	**13**
Web-Google [88]	855,802	8,582,704	10	6.30	24
YouTube [103]	1,134,890	5,975,248	5	5.32	24
Flickr [103]	1,624,992	30,953,670	18	5.38	24
AS-Skitter [86]	1,694,616	22,188,418	13	5.08	**31**
Wikipedia-NL [10]	2,213,236	23,520,520	11	4.81	18
Orkut [103]	3,072,441	234,370,166	76	4.16	10
LiveJournal3 [103]	5,189,809	97,839,882	19	5.48	23
Hyves [128]	8,083,964	912,067,984	112	4.75	25

Table 2.1: Characteristics of the datasets: various small-world graphs.

company, in which a node represents an e-mail address and two nodes are connected if an e-mail has been sent between these two addresses. The WEB-GOOGLE dataset is a partial crawl of the world wide web [88]. The FLICKR, LIVEJOURNAL, ORKUT and YOUTUBE datasets are partial crawls of the respective online social networks, and are studied in detail in [103]. AS-SKITTER is an undirected internet topology graph created from network traceroutes, which is analyzed in detail in [86]. WIKIPEDIA-NL is a full crawl of the Dutch Wikipedia graph as present in DBpedia 3.5 [10], a community effort to extract structured information from Wikipedia. The dataset denoted by HYVES is the full friendship graph of a Dutch online social network (see Chapter 6).

2.5.2 Measurement methodology

In the following experiments we will compare the three different node selection strategies from Section 2.4.4:

- **Strategy 1**: Largest eccentricity bound difference (cf. Section 2.4.4.2)
- **Strategy 2**: Interchanging largest upper bound and smallest lower bound (cf. Section 2.4.4.3)
- **Strategy 3**: Farthest distance (see Section 2.4.4.4)

Ties are broken by taking the node with the highest degree (cf. Section 2.4.4.1). Any remaining ties are broken by picking the lexicographically first node, which is determined by the order in which the nodes are read from the input file. Thus, each of the selection strategies is deterministic.

The critical step of the algorithm is clearly one BFS, so the step of computing the actual eccentricity of one node. The number of times that a BFS is executed (which we will refer to as the number of iterations) will therefore serve as a basis of comparison of the three strategies. Note that the number of iterations that the traditional APSP algorithm from Algorithm 2.1 would perform, is one eccentricity computation for each node in the graph, so a total of n iterations. We have chosen to only implement the simple optimization strategy (see Section 2.4.6) of pruning duplicate connected subgraphs consisting of one node.

2.5.3 Results

The number of iterations for each of the three strategies is given in the second, third and fourth column of Table 2.2, where a bold value indicates the best result amongst the three strategies. The last column indicates the number of pruned nodes as well as the percentage of the total number of nodes that was pruned.

The results show that with only a few actual eccentricity computations, the algorithm is able to determine the exact diameter of the datasets, with Strategy 2 as the best-performing node selection strategy. We expect this to be because interchanging the search for low and high eccentricity nodes means that we are interchanging the selection of a node in the dense and the peripheral part of the small-world network, quickly lowering the upper bound and increasing the lower bound, respectively. We believe that Strategy 1 did not perform so well, because although this strategy gives a hint towards areas of the graph that have note been thoroughly explored, it does not give any guarantees on the size of this area (which could be very small) and the number of such areas (which could be very large). Strategy 3 appeared to perform worse because it was not able to find any low-eccentricity valued nodes that could lower the diameter upper bound.

We observed that even when no selection strategy is applied (so, by selecting candidate nodes at random), using the suggested lower and upper eccentricity bounds can help to converge on the diameter more quickly than using the APSP algorithm. For larger datasets, this number was well over $10,000$ and thus still far too time-consuming, but for CA-AstroPh 260 ± 95, for Enron 316.5 ± 142 and for Web-Google a total of $5,975 \pm 2,249$ iterations were needed (the number of iterations is averaged over 10 runs, so we also report the standard deviation) to obtain the exact diameter. This may suggest that contrary to the various selection strategies, random candidate node selection does not scale as the size of the graph increases.

We mention that for the YouTube dataset, Strategy 2 only needs 2 eccentricity computations to determine the diameter, demonstrating the best-case performance of the algorithm. It turned out that the node with the highest degree had eccentricity 12, causing nodes with bounds $[12; 24]$ to exist. One of these nodes apparently had

Dataset	**Strategy 1**	**Strategy 2**	**Strategy 3**	**Pruned nodes**
CA-AstroPh	18	**9**	63	185 (1.0%)
Enron	12	**11**	61	8,715 (25.8%)
Web-Google	20	**4**	28	91,965 (10.7%)
YouTube	**2**	**2**	**2**	399,553 (35.2%)
Flickr	10	**3**	7	553,242 (34.0%)
AS-Skitter	10	**4**	19	114,803 (6.8%)
Wikipedia-NL	21	**3**	583	947,582 (30.8%)
Orkut	357	**106**	389	27,429 (0.9%)
LiveJournal	6	**3**	14	318,378 (6.1%)
Hyves	40	**21**	44	446,258 (5.6%)

Table 2.2: Performance (number of iterations) of different node selection strategies.

an eccentricity of value 24, realizing bounds of $[24; 24]$ and terminating the algorithm after just 2 iterations.

For the ORKUT dataset, compared to the other graphs, a relatively large number of eccentricity computations was needed to determine the diameter. To further analyze this, we look at how quickly the number of candidate nodes decreases during the execution of the algorithm. Therefore we show the number of unvisited nodes and the lower and upper bounds on the diameter during the execution of the algorithm using Strategy 2 on the ORKUT dataset in Figure 2.5. After 16 computations, another 90 computations were needed to decide whether the diameter was equal to 9 or 10, and the number of nodes to be examined only decreases by 1 or 2 after each eccentricity computation. Apparently the remaining unvisited nodes are positioned in the graph in such a way that the computation of the actual eccentricity of these nodes can not be avoided using the neighboring bounds. Although in this case it takes a while to find the exact diameter, tight bounds on the diameter are quickly available, as we have narrowed down the value diameter down to either one of two values.

The last column of Table 2.2 shows how the pruning strategy is able to significantly reduce the size of the problem. This is not surprising as small-world networks typically have many low degree nodes, and it is quite likely that many of these nodes are linked to the same node, and can thus be pruned.

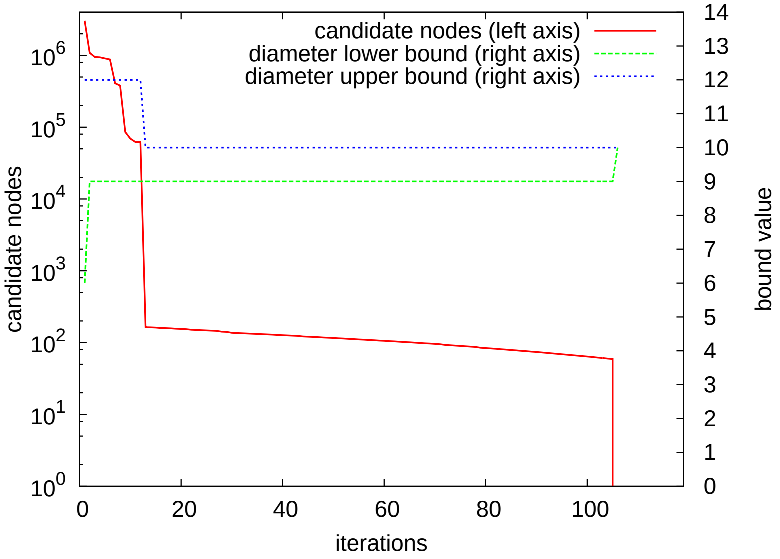

Figure 2.5: Candidate nodes (left vertical axis; logarithmic) and lower and upper bounds (right vertical axis) vs. iterations (horizontal axis) for the ORKUT dataset.

As an interesting side result it turned out that, very often, the actual eccentricity has been found for a large portion of the nodes in the graph when the algorithm has terminated, because the eccentricity lower and upper bounds of these nodes have become equal. For example, for the ORKUT dataset, $777,257$ actual eccentricity values (25%) were obtained, while only 106 values were explicitly computed. Similar numbers were observed for the other datasets. Also interesting to note is that the exact diameter that we computed for the ENRON and AS-SKITTER datasets deviates from the values of respectively 12 and 24 that were approximated in previous work.

Although not related to the performance measurement methodology, we mention that with a straightforward C++ implementation, one node eccentricity computation takes around six seconds for a dataset with 8 million nodes and 912 million edges on a standard 3.2GHz machine with 10GB of memory. This means that we are able to determine the exact diameter in a matter of seconds or minutes, which is a big improvement over the traditional APSP approach which would easily take over a year of computation time. More information on the datasets used in this chapter, the obtained diameter paths, and a simple implementation can be found at the supporting website: `www.liacs.nl/~ftakes/diameter/`. In later experiments, we ran the proposed diameter algorithm on a much larger set of graphs. The results can be found in Chapter 4. For a comparison of the proposed algorithm with related work, we refer the reader to two works that were published after the original article on which this chapter is based [36, 99].

2.6 GPU parallelism

A well-known technique to improve the performance of almost any algorithm, is to introduce parallelism in (parts of) the computation. With dual, quad and octa-core CPUs available in standard desktop machines, CPU parallelism can be an excellent way to reduce the runtime of an algorithm by performing certain operations in parallel. Nowadays, graphics processing units (GPUs) have even hundreds of cores, suggesting a much higher potential for exploiting parallelism compared to the CPU. However, whereas with a CPU algorithm it is often possible to obtain a speedup equal to the number of cores, using the GPU the speedup is usually much lower than the number of cores, as the architecture of the GPU is typically more “exotic” compared to what a regular CPU algorithm would expect. This section, which is largely based on work [41] together with Giso Dal, briefly summarizes to what extent GPU parallelism using the Nvidia CUDA framework can be applied to the BOUNDINGDIAMETERS algorithm.

Parallelizing an existing sequential algorithm is not always trivial, and involves carefully selecting procedures within the algorithm that can be parallelized. In case of the BoundingDiameters algorithm discussed in this chapter, it turned out that 97% of the running time (the performance of GPU algorithms is usually measured in seconds instead of iterations) is spent on computing eccentricity values (line 10 of Algorithm 2.2). Therefore it makes sense to optimize the eccentricity computation (so, one SSP run or one BFS). In general the specific architecture of the considered GPU should be carefully taken into account when designing data structures and algorithms that are to be used on a GPU. Extensive research on parallelizing graph traversal on a GPU has been done, and it is clear that compared to sequential CPU algorithms, GPU parallelism introduces new challenges with respect to for example shared memory and synchronization [57, 101].

For the considered NVIDIA GPU (Fermi, compute 2.0), this means that the data structure used to store the frontier of the BFS should be optimized for the specific access pattern for which the GPU memory achieves the best performance. A straightforward GPU implementation of the eccentricity function, where a thread is assigned to each node in the frontier of the BFS, already results in a speedup factor up to 12× on various large real-world graphs. The speedup appears to be somewhat dependent on the properties of the considered graph. Most notably, it appears to be influenced by the relation between the diameter and the effective diameter. A clear bottleneck of the standard (thread-based) GPU algorithm is the size of adjacency lists and the number of vertices in the frontier of the BFS. This problem can be overcome by looking at the number of nodes in the frontier at each step of the BFS. If this number is above a certain threshold, a different approach can be used, which utilizes a so-called “warp” of 32 parallel threads that processes the longer adjacency list more efficiently. Thus, a choice is made between either using one thread per adjacency list, or using multiple threads per list (but processing fewer lists in parallel). The resulting hybrid approach, which picks a different eccentricity algorithm (thread-based or warp-based) depending on the stage of the BFS, is able to realize a speedup of up to 21× compared to the sequential CPU algorithm. For a more detailed description of these GPU algorithms and experimental results, the reader is referred to [41].

2.7 Conclusion

We have shown that the proposed algorithm, BoundingDiameters, is able to efficiently determine the exact diameter of small-world networks, making use of lower and upper bounds on the eccentricity of the nodes and on the diameter itself. A proper selection strategy allows the algorithm to exploit the characteristic properties of small-world networks. Moreover, we have outlined a pruning strategy which reduces the size of the problem. We have also shown that even when the diameter is not found very quickly, very tight bounds on the diameter are available after only a few iterations.

In future work we will investigate if the proposed algorithm can be used to determine the radius (minimum eccentricity value over all nodes) of a graph. We furthermore want to see if we can obtain the eccentricity of all nodes in the network, allowing the study of the exact eccentricity distribution of a graph. These two issues will be addressed in Chapter 4 and Chapter 3, respectively.

It may also be interesting to look at the problem of determining the diameter of the strongly connected component of a directed graph, and that of weighted graphs. In line with results presented in related work [39], we expect that the bounding approach will also work well on weighted graphs, as the diversity in edge weights and thus path lengths will undoubtedly influence the difference in eccentricity values and speed up the convergence of lower and upper eccentricity bounds and therewith the diameter bounds. In preliminary experiments we see that by using only the lower diameter bounds, significant improvements over the APSP algorithm can already be observed. Following up on the GPU implementation of the proposed algorithm briefly discussed in Section 2.6, work can still be done on parallelization using the CPU or a combination of the CPU and GPU. Last but not least, we hope to investigate how the exact diameter of small-world networks behaves over time, and how the algorithm can be adjusted to adapt to changes in the network, i.e., the addition and deletion of nodes and links.

3

Computing the Eccentricity Distribution of Large Graphs

The eccentricity of a node in a graph is defined as the length of a longest shortest path starting at that node. The eccentricity distribution over all nodes is a relevant descriptive property of the graph, and its extreme values allow the derivation of measures such as the radius and diameter of the graph. This chapter describes two new methods for computing the eccentricity distribution of large graphs such as social networks, biological networks, webgraphs and routing networks. We first propose an exact algorithm based on eccentricity lower and upper bounds that is significantly faster than the straightforward algorithm when computing both the extreme values of the distribution as well as the eccentricity distribution as a whole. The second algorithm that we describe is a hybrid strategy that combines the exact approach with an efficient sampling technique in order to obtain an even larger speedup on the computation of the entire eccentricity distribution. We perform a set of experiments on a number of large graphs in order to measure and compare the performance of the proposed algorithms, and demonstrate how we can efficiently compute the eccentricity distribution of various large real-world graphs. This chapter is based on:

- F. W. Takes and W. A. Kosters. Computing the eccentricity distribution of large graphs. *Algorithms*, 6(1):100–118, 2013

3.1 Introduction

There exist all kinds of interesting properties that better describe the relationships between the objects in a dataset modeled by a graph. One of these properties is the *eccentricity distribution*, which indicates the distribution of the eccentricity over all nodes, where the *eccentricity* of a node refers to the length of a longest shortest path starting at that node. This distribution differs from properties such as the distance distribution in the sense that eccentricity can be seen as a more "extreme" measure of distance. It also differs from indicators such as the degree distribution in the sense that determining the eccentricity of every node in the graph is computationally expensive: the traditional method computes the eccentricity of each node by running an All Pairs Shortest Path (APSP) algorithm, requiring $O(mn)$ time for a graph with n nodes and m edges. Unfortunately, this approach is too time-consuming if we consider large graphs with possibly millions of nodes.

The aforementioned complexity issues are frequently solved by determining the eccentricity of a random subset of the nodes in the original graph, and then deriving the eccentricity distribution from the obtained values [118]. While such an estimate may seem reasonable when the goal is to determine the overall average eccentricity value, we will show that this technique does not perform well when the actual *extreme values* of the distribution are of relevance. The nodes with the highest eccentricity values realize the diameter of the graph and form the so-called graph periphery, whereas the nodes with the lowest values realize the radius and form the center of the graph. Finding exactly these nodes can be useful within various application areas.

In routing networks, for example, it is interesting to know exactly which nodes form the periphery of the network and thus have the highest worst-case response time to any other device [96]. Also, when (routing) networks are modeled, for example for research purposes, it is important to measure the exact eccentricity distribution so that the model can be evaluated by comparing it with the distribution of real routing networks [98]. The eccentricity also plays a role in biological networks [67], for example in networks that model some biological system. There, proteins (nodes in the network) that have a low eccentricity value are easily functionally reachable by other components of the network [111]. The diameter, defined as the length of a longest shortest path, is the most frequently studied eccentricity-based measure, and efficient algorithms for its computation have been discussed in Chapter 2 of this thesis.

Generally speaking, eccentricity can be seen as an extreme measure of centrality, i.e., the relative importance of a node in a graph. Eccentricity centrality [15], centroid centrality [23] and graph centrality [26] have been suggested as centrality measures based on the eccentricity of a node. Compared to other measures such as closeness centrality, the main difference is that a node with a very low eccentricity value is relatively close to *every* other node, whereas a node with a low closeness centrality value is close to all the other nodes *on average*.

In this chapter we first discuss an algorithm based on eccentricity lower and upper bounds for determining the *exact* eccentricity of every node of a graph. We also present a useful pruning strategy, and show how the proposed method significantly improves upon the traditional APSP-based algorithm. To realize an even larger speedup, we propose to incorporate a *sampling* technique on a specific set of nodes in the graph which allows us to obtain the eccentricity distribution much faster, while still ensuring a low error on the full eccentricity distribution and an exact result for the eccentricity-based graph properties such as the radius and diameter.

The rest of this chapter is organized as follows. We consider some notation and formulate the main problems addressed in this chapter in Section 3.2, after which we cover related work in Section 3.3. Section 3.4 describes an exact algorithm for determining the eccentricity distribution. In Section 3.5, we explain how sampling can be incorporated. Results of applying the methods to various large graphs are presented in Section 3.6, and finally Section 3.7 summarizes the chapter and offers suggestions for future work.

3.2 Preliminaries

We consider a graph $G = (V, E)$, where V is the set of $|V| = n$ vertices (nodes) and $E \subseteq V \times V$ is the set of $|E| = m$ edges (also called links). The distance $d(v, w)$ between two nodes $v, w \in V$ is defined as the length of a shortest path from v to w, i.e., the minimum number of edges that have to be traversed to get from v to w. We assume that graphs are *undirected*, meaning that $(v, w) \in E$ iff $(w, v) \in E$ and thus $d(v, w) = d(w, v)$ for all $v, w \in V$. Note that each edge (v, w) is thus included twice in E: once as a link from v to w and once as a link from w to v. We will also assume that G is connected, meaning that $d(v, w)$ is always finite. Furthermore it is assumed that there are no parallel edges and no loops linking a node to itself. The neighborhood $N(v)$ of a node v is defined as the set of all nodes connected to v via an edge: $N(v) = \{w \in V \mid (v, w) \in E\}$. The degree $deg(v)$ of a node v can then be defined as the number of nodes connected to that node, i.e., its neighborhood size: $deg(v) = |N(v)|$.

The *eccentricity* $e(v)$ of a node $v \in V$ is defined as the length of a longest shortest path from v to any other node: $e(v) = \max_{w \in V} d(v, w)$. The *eccentricity distribution* counts the frequency $f(x)$ of each eccentricity value x, and can easily be derived when the eccentricity of each node in the graph is known. The *relative eccentricity* *distribution* lists for each eccentricity value x its relative frequency $F(x) = f(x)/n$, normalizing for the number of nodes in the graph. Figure 3.1 shows the relative eccentricity distributions of a number of large graphs (for a detailed description of these datasets, see Section 3.6.1).

Various other measures can be derived using the notion of eccentricity, such as the *average eccentricity* $\overline{e}(G)$ of a graph G, defined as $\overline{e}(G) = \frac{1}{n}\sum_{v \in V} e(v)$. Another related measure is the *diameter* $D(G)$ of a graph, which is is defined as the maximum eccentricity over all nodes in the graph: $D(G) = \max_{v \in V} e(v)$. Similarly, we define the *radius* $R(G)$ of a graph as the minimum eccentricity over all nodes in the graph: $R(G) = \min_{v \in V} e(v)$. The graph center $C(G)$ refers to the set of nodes with an eccentricity equal to the radius, $C(G) = \{v \in V \mid e(v) = R(G)\}$. Similarly, the graph periphery $P(G)$ is defined as the set of nodes with an eccentricity value equal to the diameter of the graph: $P(G) = \{v \in V \mid e(v) = D(G)\}$. For ease of notation, we will often denote these measures as variables $\overline{e}$, D, R, C and P. Each of the metrics explained above can be derived when the eccentricity of all nodes is known. Figure 3.3 shows an example of a graph that explains the different measures covered in this section. In Figure 3.2, a larger graph in which the node color corresponds to the eccentricity value is shown.

Computing the eccentricity of one node can be done by running Dijkstra's algorithm, and returning the largest distance found (i.e., the distance to a node fur-

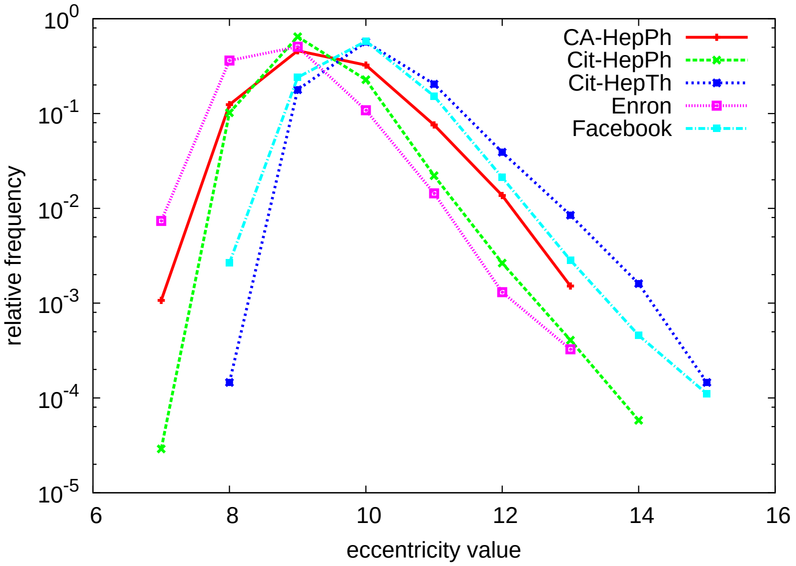

Figure 3.1: Relative eccentricity distributions of various large graphs.

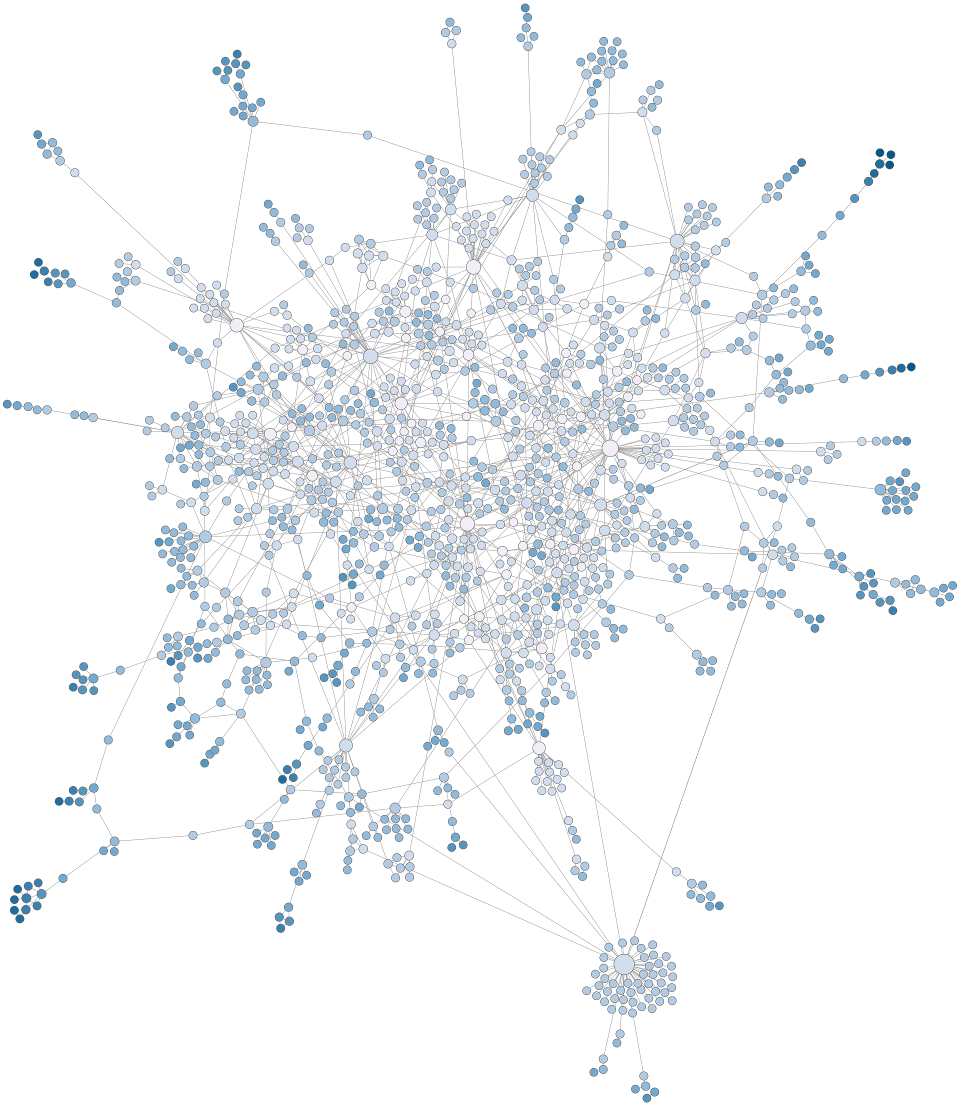

Figure 3.2: The YEAST graph, consisting of 1458 nodes and 1948 undirected edges. The color of a node represents its eccentricity value in the range from 11 to 19 from low (lighter) to high (darker) and the size of a node is proportional to its degree. Visualized using the ForceAtlas2 algorithm in Gephi (`http://gephi.org`).

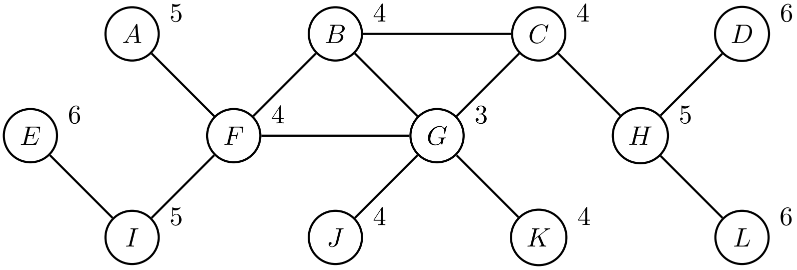

Figure 3.3: Toy graph consisting of 12 nodes and 14 edges. The number next to a node denotes its eccentricity value. The graph has average eccentricity 4.67, radius 3 realized by center node G, and diameter 6 realized by periphery nodes D, E and L.

thest away from the starting node). Because we only consider unweighted graphs and thus, starting from the current node, can simply explore the neighboring nodes in level-order, computing the eccentricity of one node can be done in $O(m)$ time. The process of determining the eccentricity of one node v is denoted by ECCENTRICITY(v) in Algorithm 3.1, representing one Breadth First Search (BFS) starting at node v.

Algorithm 3.1 simply computes the eccentricity for each of the n nodes, resulting in an overall complexity of $O(mn)$ to determine the eccentricity of every node in the graph. Clearly, in graphs with millions of nodes and possibly hundreds of millions of edges, this approach is too time-consuming. The rest of this chapter describes more efficient approaches for determining the eccentricity distribution, where we are interested in two things:

- The (relative) eccentricity distribution as a whole.
- Finding the extreme values of the eccentricity distribution, i.e., the radius and diameter, as well as derived measures such as the center and periphery.

We will address these issues by answering the following two questions:

Algorithm 3.1 NAIVEECCENTRICITIES

```
Input: Graph $G$
Output: List $e$, containing $e(v)$ for all $v \in V$
for $v \in V$ do
    $e[v] \leftarrow$ ECCENTRICITY($v$)
end for
return $e$
```

- How can we obtain the *exact* eccentricity distribution by efficiently computing the exact value of $e(v)$ for all nodes $v \in V$? (Section 3.4)
- How can we obtain an *accurate approximation* of the eccentricity distribution by using a sampling technique? (Section 3.5)

3.3 Related work

Work on the eccentricity distribution dates back to at least 1975, when the term "eccentric sequence" was used to denote the sequence that counts the frequency of each eccentricity value [90]. The facility location problem was suggested as an example of the usefulness of eccentricity as a measure of centrality. When considering the placement of emergency facilities such as a hospital or fire station, assuming the map is modeled as a graph, the node with the lowest eccentricity value might be a good location for such a facility. In other situations, for example for the placement of a shopping center, a related measure called closeness centrality, defined as the *average* distance from a particular node v to every other node ($\frac{1}{n-1}\sum_{w \in V} d(v, w)$), is more suitable. Generally speaking, the eccentricity is a relevant measure when some strict criterion (the firetruck has to be able to reach *every* location within ten minutes) has to be met [56]. An application of eccentricity as a measure on a larger scale is the network routing graph, where the eccentricity of a node says something about the worst-case response time between one machine and *all* other machines [98].

The most well-known eccentricity-based measure is the diameter, which has been extensively investigated in Chapter 2 and in [37, 95]. Several measures related to the eccentricity and diameter have also been considered. Kang et al. [68] study the effective radius, which they define as the 90th-percentile of all the shortest distances from a node. In a similar way, the effective diameter can be defined, which is shown to be decreasing over time for many large real-world graphs [86]. Each of these measures is computed by using an approximation algorithm [109] to determine the neighborhood of a node, a technique on which we will elaborate in Section 3.5.3. An overview of algorithms for approximating the radius and diameter is given in [115].

To the best of our knowledge, there are no efficient techniques that have been specifically designed to determine the exact eccentricity distribution of a graph. Obviously, the naive approach, for example as suggested in [29], is too time-consuming. Efficient approaches for solving the APSP problem (which makes deriving the eccentricities trivial) have been developed, for example using matrix multiplication [147]. Unfortunately, such approaches are still too complex in terms of time and memory requirements. The remainder of this chapter describes both a new exact algorithm and a new approximation algorithm to efficiently compute the eccentricity distribution.

3.4 Exact algorithm

In order to obtain an algorithm that can compute the eccentricity of all n nodes in a graph faster than simply recomputing the eccentricity n times (once for each node in the graph), we have two options:

1. Reduce the size of the graph to speed up one eccentricity computation.
2. Reduce the total number of eccentricity computations.

In this section we propose to use lower and upper bounds on the eccentricity, a strategy that accommodates the second type of speedup. We will also discuss a pruning strategy that helps to reduce both the number of nodes that have to be investigated, as well as the size of the graph.

3.4.1 Eccentricity bounds

We propose to use the following bounds on the eccentricity of all nodes $w \in V$:

Observation 3.1 ***Node eccentricity bounds***
If a node $v \in V$ has eccentricity $e(v)$, then for all nodes $w \in V$ we have:

$$\max(e(v) - d(v,w), d(v,w)) \leq e(w) \leq e(v) + d(v,w)$$

For an explanation of these bounds, we refer the reader to Section 2.2, in which the same observation was used to determine which nodes can contribute to the process of computing the diameter of a graph. We employ an algorithm similar to what is proposed in Chapter 2, but this time using the eccentricity bounds to compute the full eccentricity distribution of the graph. The approach is outlined in Algorithm 3.2.

First, the candidate set W and the lower and upper eccentricity bounds are initialized (lines 3–7). In the main loop of the algorithm, a node v is repeatedly selected (line 9) from W, its eccentricity is determined (line 10), and finally all candidate nodes are updated (lines 11–18) according to Observation 3.1. Note that the value of $d(v,w)$ which is used in the updating process does not have to be computed, as it is already known because it was computed for all w during the computation of the eccentricity of v. If the lower and upper eccentricity bounds for a node have become equal, then the eccentricity of that node has been derived and it is removed from W (lines 14–17). Algorithm 3.2 returns a list containing the exact eccentricity value of each node. Counting the number of occurrences of each eccentricity value results in the eccentricity distribution.

An overview of possible selection strategies for the function SELECTFROM can be found in Section 2.4.4. In line with results presented in Chapter 2, we found that

Algorithm 3.2 BoundingEccentricities

```
Input: Graph $G$
Output: List $e$, containing $e(v)$ for all $v \in V$

$W \leftarrow V$
for $w \in W$ do
    $e_\ell[w] \leftarrow -\infty$
    $e_u[w] \leftarrow +\infty$
end for

while $W \neq \emptyset$ do
    $v \leftarrow$ SelectFrom($W$)
    $e[v] \leftarrow$ Eccentricity($v$)

    for $w \in W$ do
        $e_\ell[w] \leftarrow \max(e_\ell[w], \max(e[v] - d(v,w), d(v,w)))$
        $e_u[w] \leftarrow \min(e_u[w], e[v] + d(v,w))$
        if $(e_\ell[w] = e_u[w])$ then
            $e[w] \leftarrow e_\ell[w]$
            $W \leftarrow W - \{w\}$
        end if
    end for
end while

return $e$
```

when determining the eccentricity distribution, interchanging the selection of a node with a small lower bound and a node with a large upper bound, breaking ties by taking a node with the highest degree, yielded by far the best results. As described in Section 2.4.3, examples of graphs in which this algorithm would definitely not work are complete graphs and circle-shaped graphs. However, most real-world graphs adhere to the small-world property [71], and in these graphs the eccentricity values are sufficiently diverse so that the proposed eccentricity lower and upper bounds can effectively be utilized.

3.4.2 Example run

For an example run of the proposed algorithm, consider the problem of determining the eccentricities of the nodes of the toy graph from Figure 3.3. We will denote the lower and upper eccentricity bounds $e_\ell(v)$ and $e_u(v)$ of a node v by $[e_\ell(v); e_u(v)]$. If we compute the eccentricity of node G, which is 3, then we can derive bounds $[2; 4]$

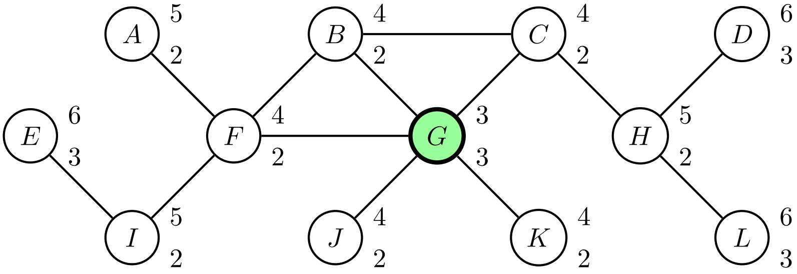

Figure 3.4: Eccentricity bounds (lower and upper bound respectively below and above the node) of the toy graph in Figure 3.3 after computing the eccentricity of node G.

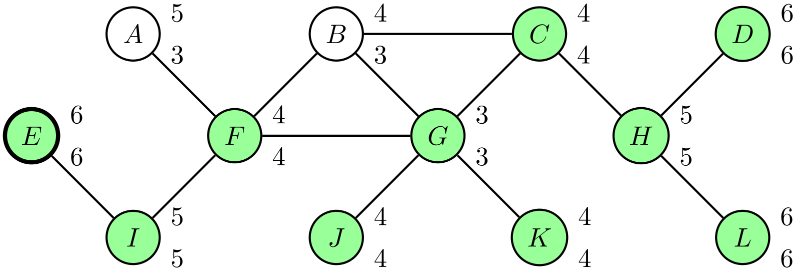

Figure 3.5: Eccentricity bounds of the toy graph in Figure 3.3 after subsequently computing the eccentricity of node G and E.

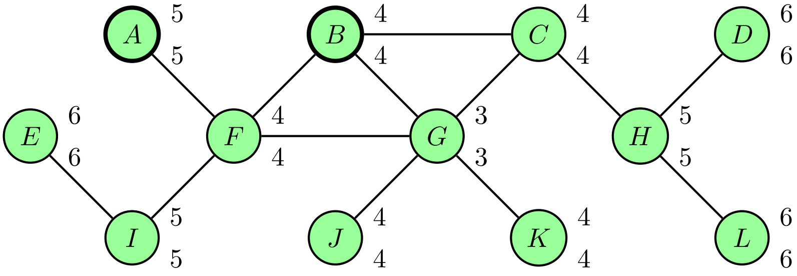

Figure 3.6: Eccentricity bounds of the toy graph in Figure 3.3 after subsequently computing the eccentricity of node G, E and A.

for the nodes at distance 1 (B, C, F, J and K), $[2; 5]$ for the nodes at distance 2 (A, H and I) and $[3; 6]$ for the nodes at distance 3 (D, E and L), as depicted in Figure 3.4. If we then compute the eccentricity of node E, which is 6, we derive bounds $[5; 7]$ for node I, $[4; 8]$ for node F, $[3; 9]$ for nodes A, B and G, $[4; 10]$ for nodes C, J and K, $[5; 11]$ for node H, and $[6; 12]$ for nodes D and L. If we combine these bounds for each of the nodes cf. lines 12-13 of Algorithm 3.2, then we find that lower and upper bounds for a large number of nodes have become equal: $[4; 4]$ for C, F, J and K, $[5; 5]$ for H and I, and $[6; 6]$ for D and L, as shown in Figure 3.5. Finally, computing the eccentricity of nodes A and B results in a total of 4 BFSes to compute the complete eccentricity distribution (Figure 3.6), which is a speedup of 3 compared to the naive algorithm, which would simply compute the eccentricity for all 12 nodes in the graph.

To give a first idea of the performance of the algorithm on larger real-world graphs, Figure 3.7 shows the number of iterations (vertical axis) that are needed to compute the eccentricity of all nodes with given eccentricity value (horizontal axis) for a number of large graphs (for a description of the datasets, see Section 3.6.1). We can clearly see that especially for the extreme values of the eccentricity distribution, very few iterations are needed to compute all of these eccentricity values, whereas many more iterations (though still much less than n) are needed to derive the values in between the extreme values.

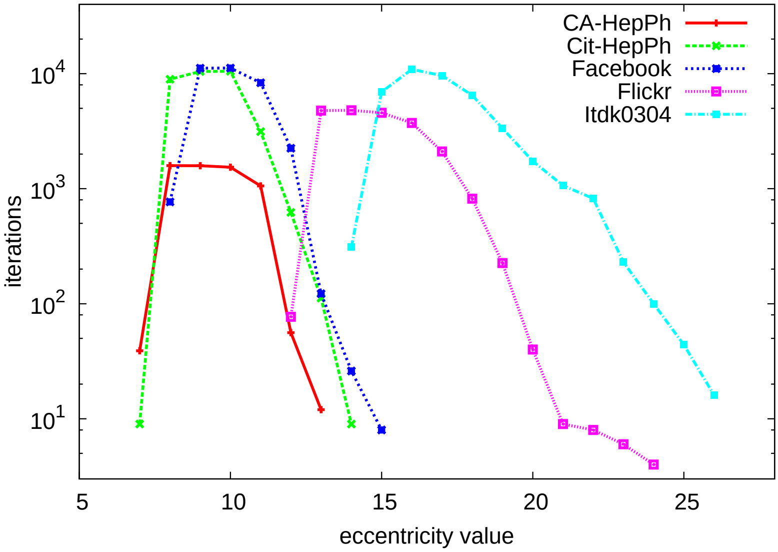

Figure 3.7: Eccentricity values (horizontal axis) vs. number of iterations to compute the eccentricity of all nodes with this eccentricity value (vertical axis, logarithmic).

3.4.3 Pruning

In this subsection we introduce a pruning strategy that is somewhat based on the pruning step introduced in Section 2.4.6. The pruning strategy is based on the following observation:

Observation 3.2 *Assume that $n > 2$. For a given $v \in V$, all nodes $w \in N(v)$ with $deg(w) = 1$ have $e(w) = e(v) + 1$.*

Node w is only connected to node v, and will thus need node v to reach every other node in the graph. If node v can do this in $e(v)$ steps, then node w can do this is in exactly $e(v) + 1$ steps. The restriction $n > 2$ on the graph size excludes the case in which the graph consists of v and w only.

All interesting real-world graphs have a lot more than two nodes, making Observation 3.2 applicable in the proposed algorithm. Observation 3.2 can be beneficial in two ways. First, when computing the eccentricity of a single node, the pruned nodes can be ignored in the shortest path algorithm. Second, when the eccentricity of a node v has been computed or derived (line 10 or lines 14–17) of Algorithm 3.2), and this node has adjacent nodes w with $deg(w) = 1$, then the eccentricity of these nodes w can be set to $e(w) = e(v) + 1$.

In Figure 3.3, node G has two neighbors with degree one, namely J and K. According to Observation 3.2, the eccentricity values of these two nodes are equal to $e(J) = e(K) = e(G) + 1 = 4$. The same argument holds for nodes D and L with respect to node H. We expect that Observation 3.2 can be applied quite often, as many of the graphs that are nowadays studied have a power law degree distribution [46], meaning that there are many nodes with a very low degree (such as a degree of 1). Furthermore, there is no significant additional computation time involved in identifying prunable nodes.

The pruning strategy described in this section is conceptually similar to the pruning strategy used in Section 2.4.6 of Chapter 2. The major difference is that there, pruned nodes are not useful for computing the diameter and can be completely removed from the graph, whereas for computing the eccentricity distribution, the actual nodes that are pruned are relevant when the final eccentricity distribution is derived from the list of eccentricity values, and should thus not be removed from the node set.

3.5 Approximation algorithms

The use of sampling to determine the eccentricity distribution will be discussed in Section 3.5.1. Sampling is a technique in which a subset of the original dataset is evaluated in order to estimate (the distribution of) characteristics of the (elements in the) complete dataset, with faster computation as one of the main advantages. We also take into account situations where we not only want to estimate the distribution of the eccentricity values, but also want to assess some parts of it (the extreme values) with high reliability. For that purpose we propose a hybrid technique to determine the eccentricity distribution that combines the exact approach from Section 3.4 with non-random sampling in Section 3.5.2. Finally in Section 3.5.3 we consider an adaptation of an existing approximation approach from literature.

3.5.1 Random node selection

If we want to apply sampling to the naive algorithm for obtaining the eccentricity distribution, we could choose to evaluate only a subset of size n' of the original n nodes (clearly, here $0 < n' < n$), and multiply the values of the sampled eccentricity distribution by a factor n/n' to get an idea of the real eccentricity distribution, a process referred to as *random node selection*.

Indeed, taking only a subset of the nodes would clearly speed up the computation of the eccentricity distribution and realize the second type of speedup discussed in Section 3.4: reducing the total number of eccentricity computations. The trade-off here is that we no longer obtain the exact eccentricity distribution, but only get an approximation. The main question is then whether or not this approximation is representative of the original distribution. The effectiveness of sampling by random node selection with respect to various graph properties has been demonstrated in [84]. Using similar arguments as presented in [38, 45], the absolute error can be assessed; indeed, when the chosen sampling subset is sufficiently large, the error is effectively bounded.

However, there are situations where we are not interested in minimizing the absolute error, but in minimizing the relative standard deviation (i.e., the absolute standard deviation divided by the mean) of the distribution of a particular eccentricity value. This especially makes sense when each eccentricity value is of equal importance, which might be the case when the extreme values of the distribution that are realizing the radius and diameter are of relevance. Let us consider an example. The real exact eccentricity distribution over all nodes in the ENRON graph [73] is listed in Table 3.1.

$e(v)$	7	8	9	10	11	12	13
$f(e(v))$	248	12,210	17,051	3,647	485	44	11

Table 3.1: Eccentricity distribution of the ENRON graph.

Figure 3.8 shows the relative standard deviation for each eccentricity value for different sample sizes (1%, 2%, 5%, 10%, and 20%). For each sample size, we took 100 random samples to average the result. We note that low relative standard deviations can be observed for the more common eccentricity values (in this case, 8, 9 and 10). However, the standard deviation is quite large for the extreme values, meaning that on many occasions, the sample did not correctly reflect the frequency of that particular eccentricity value. Indeed, if only 11 out of 33,696 nodes (0.03%) have a particular eccentricity value (in this case, a value of 13), we need a sample size of at least $100\%/11 = 9\%$ if we want to expect just one node with that particular eccentricity value.

Figure 3.8 shows that even with a large sample size of 20%, the standard deviation for the extreme value of 13 is very high (0.63). For sample sizes of 1% or 2%, the eccentricity value of 13, that exists in the real distribution, was never even found. Similar events were observed for the other datasets, which is not surprising: the eccentricity distributions are tailed on the extremes, and these tails are likely to be left out in a sample. So clearly sampling does not suffice when extreme values of the eccentricity distribution have to be found, because such extreme values are simply too rare. However, for the more common eccentricity values, errors are very low, even for small sample sizes.

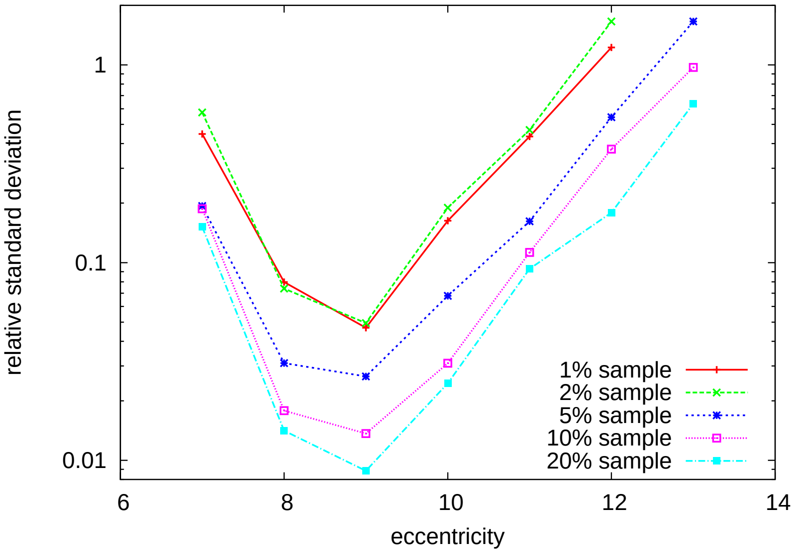

Figure 3.8: Relative standard deviation (vertical axis) of eccentricity values (horizontal axis) for different random sample sizes of the ENRON dataset.

3.5.2 Hybrid algorithm

From Figure 3.7 and Figure 3.8 we can conclude that the exact approach is able to quickly derive the more extreme values of the eccentricity distribution, whereas a sampling technique is able to approximate the values in between the extremes with a very low error. Therefore we propose to combine the two approaches in order to quickly converge to an accurate estimation of the eccentricity distribution using a *hybrid* approach. We mention that the proposed technique assumes that the eccentricity distribution has a somewhat unimodal shape, which is the case for all real-world graphs that we have investigated.

The *sampling window* consists of bounding variables ℓ and r that denote between which eccentricity values the algorithm is going to use the sampling technique. The sampling window obviously depends on the distribution itself, which is not known until the exact algorithm has finished. Therefore we set the value of ℓ and r dynamically as outlined on lines 19–20 of Algorithm 3.3, which we will call BOUNDINGECCENTRICITIESLR. The main difference with respect to Algorithm 3.2 is a change in the stopping criterion in line 8. This means that the algorithm should now stop when there are no more candidates that are potentially part of the center or the periphery of the graph. Note that these bounds apply to the candidate set W, but (via the values of ℓ and r) we use information about all nodes V, ensuring that at least the center and periphery are known before the exact phase is terminated. When the exact algorithm has done its job, it can tighten ℓ and r even further, based on the eccentricity of the remaining candidate nodes, as outlined on lines 22–23. This can be beneficial when the center is much harder to find than the periphery (or the other way around), in which case the distribution of more eccentricity values may already have been computed exactly, resulting in the advantage that for these eccentricity values no sampling has to be done. After this, ℓ and r become static, and the rest of the eccentricity distribution will be derived by using the sampling approach outlined in Algorithm 3.4.

Compared to Algorithm 3.2, the difference in terms of input is that the hybrid algorithm given in Algorithm 3.4 takes as input both the original graph and a sampling rate q between 0 and 1, where $q = 0.10$ means that 10% of the nodes have to be sampled. In Section 3.6 we will perform experiments to determine how q should be set. The resulting list f holds the eccentricity distribution and is initialized based on the exact eccentricity values e' for values outside the sampling window (line 7). Nodes for which the exact eccentricity is not yet known are added to set Z (line 9). In lines 12–17, the eccentricity of a total of $n \cdot q$ random nodes are computed, and if this value lies within the specified window, the frequency of this eccentricity value is increased proportionally to the sample size. Finally, the eccentricity distribution f is returned.

To get a first idea of how the hybrid algorithm works, we look at Figure 3.9 of

Algorithm 3.3 BoundingEccentricitiesLR (adjustment of Algorithm 3.2)

```
Input: Graph $G$ and a reference to variables $\ell$ and $r$
Output: List $e$, containing $e(v)$ for values of $e(v)$ outside $[\ell; r]$
$W \leftarrow V$    $\ell \leftarrow 0$    $r \leftarrow \infty$
... // original BoundingEccentricities algorithm
while $\{w \in W \mid e_\ell[w] = \ell$ or $e_u[w] = r\} \neq \emptyset$ do
    ... // original BoundingEccentricities algorithm
    $\ell = \min_{v \in V} e_\ell[v]$
    $r = \max_{v \in V} e_u[v]$
end while
$\ell = \min_{v \in W} e_\ell[v] - 1$
$r = \max_{v \in W} e_u[v] + 1.$
return $e$
```

Algorithm 3.4 HybridEccentricities

```
Input: Graph $G$ and sampling rate $q$
Output: List $f$, containing the eccentricity distribution of $G$, initialized to 0
$e' \leftarrow$ BoundingEccentricitiesLR$(G, \ell, r)$
$Z \leftarrow \emptyset$
for $v \in V$ do
    if $e'[v] \neq 0$ and $(e'[v] \leq \ell$ or $e'[v] \geq r)$ then
        $f[e'[v]] \leftarrow f[e'[v]] + 1$
    else
        $Z \leftarrow Z \cup \{v\}$
    end if
end for
for $i \leftarrow 1$ to $n \cdot q$ do
    $v \leftarrow$ RandomFrom$(Z)$
    $e[v] \leftarrow$ Eccentricity$(v)$
    $f[e[v]] \leftarrow f[e[v]] + (1/q)$
    $Z \leftarrow Z - \{v\}$
end for
return $f$
```

the CA-HEPPH graph with radius $R = 7$ and diameter $D = 13$. The figure shows for each eccentricity value on the left axis the number of exact iterations to compute all eccentricities of that value (just like Figure 3.7) and the relative standard deviation when sampling (just like Figure 3.8) on the right axis. Clearly, if we exactly compute the eccentricity values 7, 12 and 13, and use sampling within the window $[8; 11]$, then the region below the horizontal line in Figure 3.9 indicates how we obtain the eccentricity distribution of the CA-HEPPH graph with very low errors, while not using many exact BFSes for the extreme values. Note that we could also have chosen $[8; 12]$ as a sampling window, as the sampling error for an eccentricity value of 12 is still rather low. We will further analyze the performance of the hybrid approach using a larger set of graphs in Section 3.6.

3.5.3 Neighborhood approximation

Algorithms for efficiently computing the neighborhood function have been introduced in [18, 68, 109]. These algorithms can be adjusted as follows to also approximate the eccentricities. For an integer $h > 0$, the normalized size of the neighborhood $N_h(u)$ of a node u can be defined as:

$$N_h(u) = \frac{1}{n-1} \,|\, \{v \in V \mid 0 < d(u,v) \leq h\} \,|$$

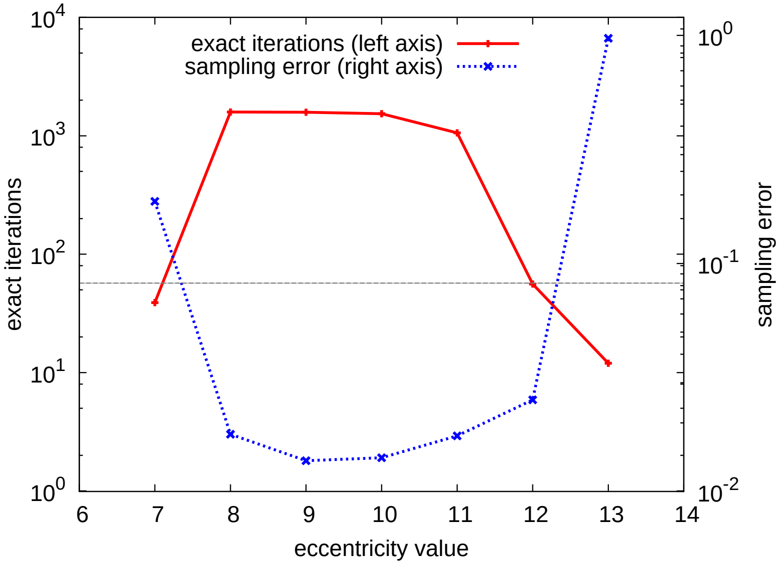

Figure 3.9: Number of exact iterations and sampling relative standard deviation for different eccentricity values of the CA-HEPPH dataset.

If we determine the value of $N_h(u)$ for increasing values of h, then the eccentricity $e(u)$ is the smallest h such that the approximated value of $N_h(u)$ is sufficiently close to 1 or does not change in successive iterations.

We have used the Approximate Neighborhood Function (ANF) [109] algorithm by Palmer et al. in order to approximate $N_h(u)$ (using the C source code available at the author's website). Figure 3.10 shows the actual relative eccentricity distribution of the ENRON dataset, as well as the distribution that was approximated using the ANF-based approach. As ANF gives an approximate result, we averaged the result of 100 runs. We note that although the distribution shapes are very similar, ANF clearly underestimates the eccentricity values. We furthermore mention that the obtained distribution is clearly an approximation: a real eccentricity distribution would never have eccentricity values for which the smallest and largest value differ more than a factor of 2. Similar results were observed for other datasets, which leads us to believe that these approximation algorithms are not suitable for determining the eccentricity distribution, especially because they fail at assessing the width and the extreme values of the distribution.

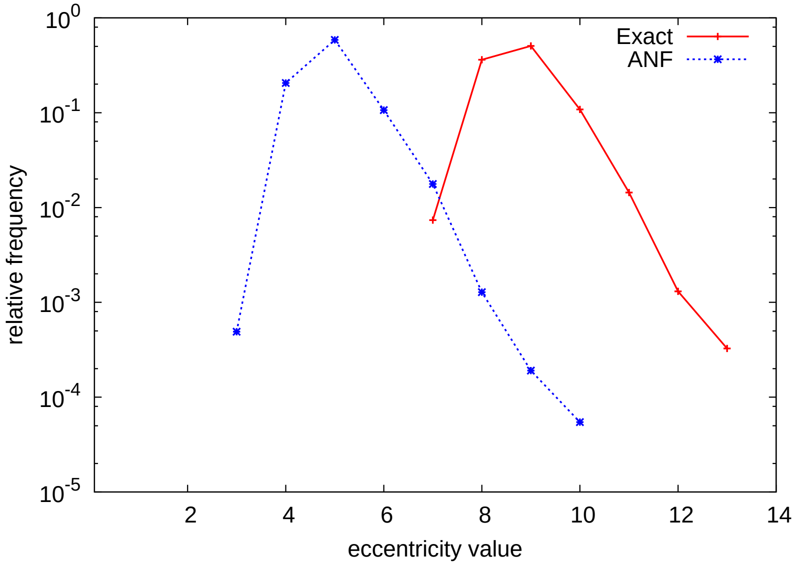

Figure 3.10: Exact relative eccentricity distribution of the ENRON dataset and an approximation using ANF.

3.6 Experiments

In this section we will use a number of large real-world datasets to assess the performance of both the exact algorithm from Section 3.4 and the hybrid algorithm from Section 3.5. The performance of the algorithms is measured by comparing the number of iterations (actual eccentricity computations by means of a BFS) and is therefore independent of the hardware and software used to implement and run the algorithm.

3.6.1 Datasets

We use a variety of graph datasets, of which the name and source are listed in the first column of Table 3.2. The set of graphs covers a broad range of real-world graphs:

- Citation networks (CIT-HEPTH, CIT-HEPPH)
- Scientific collaboration networks (CA-HEPTH, CA-HEPPH, CA-CONDMAT and DBLP20080824)
- Communication networks (ENRON, SLASHDOT)
- Peer-to-peer networks (P2P-GNUTELLA)
- Potein-protein interaction networks (YEAST, DIP20090126),
- Router topology networks (ITDK0304-RLINKS, AS-SKITTER)
- Social networks (FACEBOOK, EPINIONS, SOC-SLASHDOT, FLICKR)
- Webgraphs (WEB-STANFORD, WEB-NOTREDAME, EU-2005)

We will only consider the largest connected component of each graph, which is always the vast majority of the original graph. We also mention that some directed graphs have been interpreted as if they were undirected, and that self-edges and parallel edges are ignored. These factors may cause minor differences between the number of nodes and edges that we present here, and the numbers presented in the source papers. In Table 3.2 we also present for each dataset the exact average eccentricity $\overline{e}$, radius R, diameter D and center and periphery sizes $|C|$ and $|P|$. For a more detailed description of these graphs, we refer the reader to the source papers in which the datasets were introduced.

Looking at the exact eccentricity distributions that we were able to compute for each of the graphs, we can conclude that the distribution is not perfectly Gaussian as [97] suggests, but does appear to be unimodal. The distribution appears to be somewhat “tailed”: a positive skew is noticeable in Figure 3.1, which shows the relative eccentricity distribution of a number of graphs. The tail also becomes clear when comparing the average eccentricity value $\overline{e}$ with the radius and diameter of the graph: for every dataset, the average eccentricity is closer to the radius. For example, for the CIT-HEPTH dataset with radius 8 and diameter 15, the average eccentricity is 10.14.

3.6.2 Exact algorithm

In order to determine the performance of the exact algorithm (Algorithm 3.2), we can count the number of shortest path computations that are needed to obtain the full distribution, and compare this value to n, the number of iterations performed by the naive algorithm (Algorithm 3.1). The number of iterations performed by the exact algorithm is displayed in the third column of Table 3.3, followed by the speedup factor compared to the naive algorithm. We note that significant speedups can be observed in terms of the number of iterations, especially for datasets for which the eccentricity distribution is wide, i.e., the difference between the radius and diameter is very large.

Dataset	Nodes n	Links m	$\overline{e}$	R	D	$\lvert C\rvert$	$\lvert P\rvert$
YEAST [64]	1,458	3,896	13.28	11	19	48	4
CA-HEPTH [87]	8,638	49,612	12.53	10	18	74	4
CA-HEPPH [87]	11,204	235,238	9.40	7	13	12	17
DIP20090126 [121]	19,928	82,404	22.01	15	30	1	2
CA-CONDMAT [86]	21,363	182,572	10.58	8	15	6	11
CIT-HEPTH [86]	27,400	704,080	10.14	8	15	4	4
ENRON [73]	33,696	361,622	8.77	7	13	248	11
CIT-HEPPH [86]	34,401	841,612	9.18	7	14	1	2
SLASHDOT [52]	51,083	243,780	11.66	9	17	7	3
P2P-GNUTELLA [87]	62,561	295,754	8.94	7	11	55	118
FACEBOOK [103]	63,392	1,633,772	9.96	8	15	168	7
EPINIONS [114]	75,877	811,476	9.74	8	15	614	6
SOC-SLASHDOT [88]	82,168	1,086,761	8.91	7	13	484	3
ITDK0304 [121]	190,914	1,215,220	17.09	14	26	155	7
WEB-STANFORD [88]	255,265	3,883,852	106.49	82	164	1	3
WEB-NOTREDAME [12]	325,729	2,180,216	27.76	23	46	12	172
DBLP20080824 [121]	511,163	3,742,140	14.79	12	22	72	9
EU-2005 [121]	862,664	32,276,936	14.03	11	21	3	4
FLICKR [103]	1,624,992	30,953,670	15.03	12	24	17	3
AS-SKITTER [86]	1,694,616	22,188,418	21.22	16	31	5	2

Table 3.2: Number of nodes, links, average eccentricity, radius, diameter, center size and periphery size of the various graph datasets.

We believe that this is due to the fact that the nodes that form the periphery of the network are sufficiently eccentric to have a large influence on the more central nodes, so that their lower and upper eccentricity bounds can very quickly be fixed by the bounding technique. We mention that the performance of the exact algorithm does not appear to be directly related to the number of nodes.

The reduction in terms of the number of nodes as a result of the proposed pruning strategy (see Section 3.4.3) is displayed in the column labeled “Pruned”, and is diverse, yet sometimes very significant (anywhere between 1% and 34%). To get a better idea of the performance of the algorithm, we propose to look at the quality of both the lower and upper bounds as the exact algorithm iterates over the candidate nodes. For this we define the measure of *bound accuracy* as the percentage of bound values that are correct at that iteration:

Dataset	**Exact algorithm**			**Hybrid algorithm**			
	Pruned	**Iter.**	**Speedup**	**Exact**	**Sampled**	**Total**	**Speedup**
Yeast	399	213	**8.7**	104	483	587	3.1
CA-HepTh	351	1,055	8.2	150	350	500	**17.3**
CA-HepPh	282	1,588	7.1	57	264	321	**34.9**
Dip20090126	3,032	224	**89.0**	8	1,321	1,329	15.0
CA-CondMat	353	3,339	6.4	73	388	461	**46.3**
Cit-HepTh	140	8,104	3.4	57	444	501	**54.7**
Enron	8,715	678	**49.7**	536	145	681	49.5
Cit-HepPh	150	10,498	3.3	37	271	308	**112**
Slashdot	19,255	31	**1,648**	24	180	204	250
P2P-Gnutella	16,413	21,109	3.0	8,575	177	8,752	**7.1**
Facebook	1,075	11,185	5.7	780	168	948	**66.9**
Epinions	20,779	4,302	17.6	1,308	75	1,383	**54.9**
Soc-slashdot	14,848	1,460	56.3	990	156	1,146	**71.7**
Itdk0304	16,434	10,830	17.6	312	960	1,272	**150**
Web-Stanford	10,350	9	**28,363**	8	1,198	1,206	212
Web-NotreDame	141,178	143	2,277	94	1,381	1,475	**4,528**
Dblp20080824	22,579	42,273	12.1	150	355	505	**1,012**
EU-2005	26,507	59,751	14.4	71	1,630	1,701	**507**
Flickr	553,242	4,810	338	200	1,618	1,818	**932**
AS-Skitter	114,803	42,996	39.4	14	308	322	**5,502**

Table 3.3: Performance of the exact algorithm (Algorithm 3.2) and the hybrid algorithm (Algorithm 3.4).

$$\textit{bound accuracy} = \frac{1}{n}\ \left|\{v \in V \mid e_{real}(v) = e_{bound}(v)\}\right|$$

Here, $e_{real}(v)$ is the actual eccentricity value, and $e_{bound}(v)$ is the (lower or upper) bound that we investigate. Figure 3.11 shows the lower and upper bound accuracy for a number of datasets. We can observe that for each of the datasets, after just a few iterations, the lower bound gives a very accurate indication of the actual eccentricity values. Apparently, the majority of the iterations are spent lowering the upper bound, whereas the lower bound quickly reflects the actual eccentricity distribution. Thus, in order to get an online estimate of the distribution, we could choose to consider only the lower bound, and obtain an accuracy of around 90% after a handful of iterations.

3.6.3 Hybrid algorithm

In the second set of experiments, we have evaluated the performance of the hybrid approach that incorporates sampling for the non-extreme values of the distribution (Algorithm 3.4). We will verify the quality of the obtained distribution by using the measure of *distribution error*, defined as the sum of the absolute difference between the eccentricity counts of real relative eccentricity distribution $F(x)$ and the estimated distribution $\hat{F}(x)$:

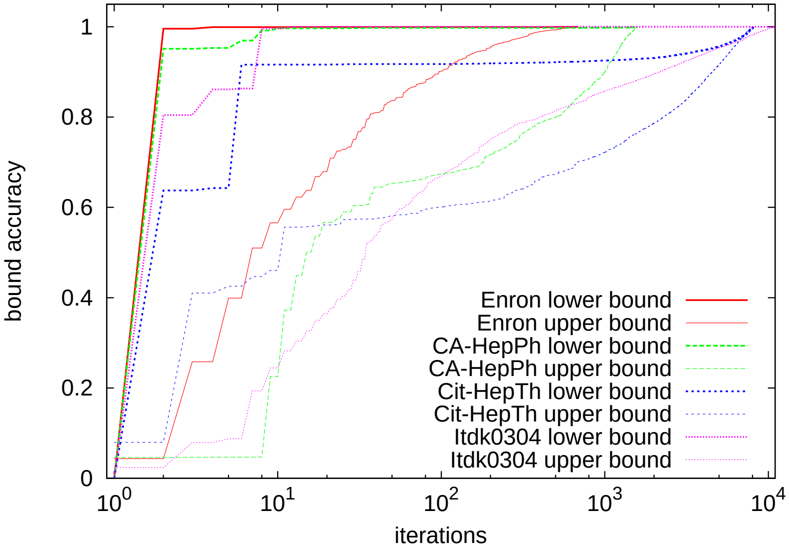

Figure 3.11: Bound accuracy vs. number of iterations for Algorithm 3.2.

$$\textit{distribution error} = \sum_{x=R}^{D} |F(x) - \hat{F}(x)|$$

Here a distribution error value of 0 indicates a perfect match between the actual and estimated relative eccentricity distributions. So for the hybrid approach, the error outside the sampling window is always equal to 0. Note that as a result of the exact approach, we have access to the real eccentricity distribution to make this error comparison. Therefore we can investigate the number of iterations that are needed to obtain an error of 0.05 (or lower) in order to determine how we should set the sampling rate q.

The fifth column of Table 3.3 shows the number of iterations needed to (exactly) compute the extreme values, which together with the column titled “Sampling” (averaged over 10 runs) results in the value denoted in column “Total”, which shows the total number of iterations needed to obtain an error of 0.05 or lower. Apparently, for large graphs (over 100,000 nodes), a sampling rate of $q = 0.10$ is more than sufficient to accurately derive the eccentricity distribution. An alternative would be to change the number of samples to a constant number, which would ensure that the number of eccentricity computations does not exceed some fixed maximum.

An advantage of the exact approach is obviously that the nodes that have a certain eccentricity value can be pointed out exactly, whereas this is only possible for the extreme values in case of the hybrid approach. The hybrid approach does however improve upon the exact algorithm in terms of computation time (number of iterations) in many cases, especially for the larger graphs (over 100,000 nodes) with relatively tight eccentricity distributions, as can be seen in the rightmost column of Table 3.3.

A bold value indicates the largest of the two speedups when comparing the hybrid approach to the exact approach. The exact approach performs better than the hybrid approach in a few cases, which we believe is due to the fact that the sampling phase should not have a too large window when the number of nodes is very small, because then a large number of iterations would be needed to obtain a representative sample of the distribution. This is especially the case for WEB-STANFORD, which has a very long tail. For this dataset, the exact algorithm performs really well, whereas the hybrid approach needs quite a few more iterations. It would be possible to determine a priori whether the exact or the hybrid algorithm should be used, as after a few BFSes the expected width of the eccentricity distribution is already known. We mention that, analogously to the performance of the exact algorithm, the performance of the hybrid algorithm does not appear to be correlated with the number of nodes.

From the experiments in this section we can generally conclude that we have developed a flexible approach for determining the eccentricity distribution of a large graph with high accuracy, while guaranteeing an exact result on the extreme values of the distribution.

3.7 Conclusion

In this chapter we have studied means of computing the eccentricity of all nodes in a graph, resulting in the (relative) eccentricity distribution of the graph. It turns out that the eccentricity distribution of real-world graphs has an unimodal shape and tends to have a positive tail. We have shown how large speedups compared to the traditional method can be achieved by considering lower and upper bounds on the eccentricity, and by applying a pruning strategy. Even when the exact algorithm does not immediately give a satisfying result, the lower bounds proposed in this chapter can serve as a reliable online estimate of the distribution as a whole.

We have also investigated the use of sampling as a means of computing the eccentricity distribution. The resulting hybrid algorithm uses the exact approach to derive the extreme values of the distribution and a sampling technique within a specific sampling window to accurately assess the values in between the extreme values. This results in an overall approach that is able to efficiently determine the eccentricity distribution with only a fraction of the number of computations required by the naive approach.

In future work we would like to investigate the extent to which eccentricity and other centrality measures are related, and when the eccentricity can be used as a meaningful centrality measure. We furthermore found that the radius, center and periphery of the graph can be derived using the exact algorithm proposed in this chapter. The question remains whether or not these measures can be computed more efficiently if the right stopping criterion is used, which we will discuss in Chapter 4. It may also be interesting to look at how the eccentricity distribution of a graph that is evolving through the addition and deletion of nodes and edges can be efficiently computed and monitored.

4

A Bounding Framework for Computing Extreme Graph Measures

In this chapter, the two algorithms for computing the exact diameter and eccentricity distribution introduced in Chapter 2 and Chapter 3 are combined and generalized so that the resulting framework can not only be used to efficiently compute the diameter and eccentricity values, but also to efficiently compute the radius, center and periphery of a graph. We will report on a number of experiments on a large set of 75 real-world graphs as well as a synthetic graph, and attempt to link the performance of the proposed framework to the various properties of the considered graphs.

4.1 Introduction

Graphs have all kinds of interesting metrics that are based on the distance between nodes. This chapter specifically considers so-called *extreme* distance measures that are based on node eccentricity (longest shortest path length from a particular node). The focus is on a unified framework for computing the exact diameter (see Chapter 2), radius, center (size) and periphery (size) (see Chapter 3) of a graph. Obviously, if we know the full distance matrix of the graph, then we can immediately derive distance extrema by simply determining and counting the lowest and highest values in the matrix. However, graphs that are nowadays studied are typically very large, and exact computation of the full distance matrix by using an All Pairs Shortest Path (APSP) like approach is impossible due to time and memory limitations.

An example of an application of extreme distance measures can be found in the well-known concept of "six degrees of separation", which is sometimes interpreted as every other person in the world (a large social network of people) being *at most* six handshakes away from each other. Some other definitions conveniently relax this statement to six handshakes *on average*, as the average node-to-node distance in many social networks is often somewhere between 4 and 6. However, if we consider the first definition, then every node should have an eccentricity value of six, and a node's degree of separation is equal to its eccentricity. For example, the degrees of separation of various movie stars in a network in which nodes represent actors and edges connect two actors that played together in a movie, are presented in [21].

Our framework for computing different extreme distance measures is based on the diameter algorithm from Chapter 2 and the exact eccentricity algorithm discussed in Chapter 3. Although the eccentricity algorithm is in theory able to derive each of the other extreme distance measures (including the diameter), there are some significant differences. Specifically, the stopping condition, the policy of when a node is discarded as a candidate, the bound updating rules and the extent to which the pruning strategy is helpful, are different for each of the considered measures. These factors also influence the performance: the number of iterations required to determine the radius of the graph is significantly less when we specifically tune the algorithm to find the radius, as compared to when we would simply discover it as a by-product of the eccentricity distribution.

For an overview of more applications of extreme distance measures, the advantages of using an exact instead of an approximation algorithm, best and worst case complexity, an example run of the bounding algorithms, the proposed pruning strategy, node selection strategies and an overview of related work, we refer the reader to Chapter 2 and Chapter 3. This chapter contributes to the previous two chapters by providing a larger number of experiments on 75 different real-world graphs consisting of up to 60 million nodes, as well as an analysis of how the algorithm's performance is related to specific properties of the graph. The strongest observed relation is then verified in a more confined experiment on a synthetic graph.

The rest of this chapter is organized as follows. We formally define the different extreme distance measures that we consider in this chapter in Section 4.2. Next, Section 4.3 explains the proposed unified bounding framework. In Section 4.4 and Section 4.5, a number of experiments is performed on respectively a large set of real-world graphs and a synthetic graph. Section 4.6 concludes the chapter.

4.2 Definitions

Given an unweighted graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges, the distance $d(u, v)$ between two nodes $u, v \in V$ is the minimum number of edges that connects nodes u and v. We assume that the graph is undirected, meaning that $(u, v) \in E$ iff $(v, u) \in E$. So, each undirected edge is included in the set of edges twice (once in each direction). The assumption is that G is connected, i.e., $d(u, v)$ is finite for all nodes u and v. Recall that we have the following definitions:

- the *eccentricity* $e(v)$ of a node v is the length of a longest shortest path starting at node v, i.e., $e(v) = \max_{w \in V} d(v, w)$.
- the *diameter* $D(G)$ is the length of a longest shortest path over all nodes and defined as $\max_{v \in V} e(v)$.
- the *radius* $R(G)$ is the length of a shortest shortest path over all nodes, and thus the minimum eccentricity over all nodes, i.e., $\min_{v \in V} e(v)$.
- the *periphery* $P(G)$ is the set of nodes for which the eccentricity is equal to the diameter, i.e., $\{v \in V \mid e(v) = D(G)\}$.
- the *center* $C(G)$ is the set of nodes for which the eccentricity is equal to the radius, i.e., $\{v \in V \mid e(v) = R(G)\}$.

The eccentricity of a node v can be determined by performing one Breadth First Search (BFS) rooted in node v, using $O(m)$ time. A naive algorithm for the four measures does this for each of the n nodes, costing a total of $O(mn)$ time, essentially performing an All Pairs Shortest Path (APSP) run. An extension to weighted graphs is trivial, as the distance function can take edge weights into account. We do not consider directed graphs, but do mention that in the directed case we can either compute the desired measure only for the strongly connected component, or apply the notion of forward and backward eccentricity as described in [99].

4.3 Framework

This section describes the proposed algorithm for computing each of the four measures described in the previous section. First, some bounds on the eccentricity are discussed, after which the general framework is outlined. Each of the measures that we compute uses this framework in a different way. A simple implementation can be found at `http://www.liacs.nl/~ftakes/bounding`.

4.3.1 Bounds

The following bounds, in case of the diameter $D(G)$ also discussed in Section 2.4.1, are important for efficiently determining extreme distance measures.

4.3.1.1 Diameter and radius bounds

Given two lists of length n of bounds $e_\ell(v)$ and $e_u(v)$ of derived lower and upper bounds of the eccentricity of each of the nodes v in the graph (cf. Section 2.4.1), we say that:

- For the diameter $D(G)$ it holds that $D_\ell(G) \leq D(G) \leq D_u(G)$ with:
  - The diameter lower bound $D_\ell(G) = \max_{v \in V} e_\ell(v)$.
  - The diameter upper bound $D_u(G) = \max_{v \in V} e_u(v)$.
- For the radius $R(G)$ it holds that $R_\ell(G) \leq R(G) \leq R_u(G)$ with:
  - The radius lower bound $R_\ell(G) = \min_{v \in V} e_\ell(v)$.
  - The radius upper bound $R_u(G) = \min_{v \in V} e_u(v)$.

4.3.1.2 Usefulness

The proposed framework keeps a set of candidate nodes that can contribute to computing the considered metric, and the following expressions precisely define this usefulness:

- For the diameter $D(G)$, a node v is useful if:
  - $e_u(v) > D_\ell(G)$: v can potentially increase the diameter lower bound.
  - $2 \cdot e_\ell(v) < D_u(G)$: v can potentially decrease the diameter upper bound.
- For the radius $R(G)$, a node v is useful if:
  - $e_\ell(v) < R_u(G)$: v can potentially decrease the radius upper bound.

  - $\lceil e_u(v)/2 \rceil > R_\ell(G)$: v can potentially increase the radius lower bound.

- For the periphery $P(G)$, a node v is useful if:
  - $e_u(v) \geq D_\ell(G)$: v can potentially realize (or increase) the diameter (lower bound) and thus belong to the periphery.
  - $2 \cdot e_\ell(v) \leq D_u(G)$: v can potentially decrease the diameter upper bound. This can be discarded when the diameter has been found.

- For the center $C(G)$, a node v is useful if:
  - $e_\ell(v) \leq R_u(G)$: v can potentially realize (or decrease) the radius (upper bound) and thus belong to the center.
  - $\lceil e_u(v)/2 \rceil > R_\ell(G)$: v can potentially increase the radius lower bound. This can be discarded when the radius has been found.

If the eccentricity bounds of a node are such that they cannot contribute to the current metric, then the eccentricity of this node does not have to be computed to determine the metric, meaning that it can be removed from the candidate set. This is the basis of our algorithm.

4.3.2 Algorithm

The main algorithm of our framework is outlined in Algorithm 4.1. Lines 3–6 initialize some bound values as well as the candidate set W, which is used in the stopping criterion (line 7) of the main loop of the algorithm.

The SELECTFROM-function (line 8) selects a node from the candidate set for which the eccentricity is going to be determined. In Section 2.4.4, a set of experiments is performed to see which selection method works best. Indeed, a good selection method is crucial for quickly converging the lower and upper bounds and thus for quickly finding the value of the considered metric. Generally, it turns out that the best performance is obtained when repeatedly selecting the node with the smallest lower bound and then the one with the largest upper bound. Any ties are broken by taking the node with the highest degree, as this node has the most potential to influence the eccentricity of neighboring nodes.

The ECCENTRICITY-function (line 9) computes the eccentricity of one node (so, it performs one full BFS from the source node). In doing so, it also conveniently finds the distance from that node to every other node. These distances are needed by the distance function d used to update the eccentricity bounds on line 11 and 12. The maximum distance found is obviously the value of the eccentricity of the node.

Algorithm 4.1 ExtremaBounding

```
 1: Input: Graph $G = (V, E)$, desired output measure $Q$
 2: Output: Value of $Q$

 3: for $v \in V$ do
 4:   $e_\ell[v] \leftarrow -\infty$    $e_u[v] \leftarrow \infty$
 5: end for
 6: $W \leftarrow V$

 7: while $W \neq \emptyset$ do
 8:   $v \leftarrow$ SelectFrom($W$)
 9:   $e[v] \leftarrow$ Eccentricity($v$)
10:   for $w \in W$ do
11:     $e_\ell[w] \leftarrow \max(e[w], e[v] - d(v,w), d(v,w))$
12:     $e_u[w] \leftarrow \min(e[w], e[v] + d(v,w))$
13:     if not IsUseful($w, Q$) then
14:       $W \leftarrow W - \{w\}$
15:     end if
16:   end for
17: end while
18:
19: return VALUE_OF($Q$)
```

The IsUseful-function (line 13) determines whether or not a node w is useful for on the one hand computing the current metric Q (either one of the measures discussed in Section 4.2), and on the other hand for tightening the lower and upper bounds of the eccentricity values. Depending on the considered metric, this function behaves according to the expressions with respect to usefulness in Section 4.3.1. Note that this is the part of the algorithm that determines which metric is actually computed. So depending on the desired output metric Q, the candidate set may shrink faster or slower, determined by whether or not the remaining nodes are useful for determining the current metric.

We note that the pruning strategy outlined in Section 2.4.6 can be applied before the algorithm is run. For the radius and diameter, this will not influence the result, but for the center and periphery size some simple book-keeping has to be done to ensure a correct result.

For a partial comparison of the proposed algorithm with related work, we refer the reader to [21, 36, 99].

4.4 Experiments on real-world graphs

This section presents an extensive number of experiments on a number of graph datasets, as well as a numeric comparison of this performance with properties of the respective graphs.

4.4.1 Datasets

For a total of 75 graph datasets, we list basic statistics such as the name of the dataset, its source (website or article), the number of nodes n, the number of links m and the average node-to-node distance $\overline{d}$ and average eccentricity $\overline{e}$ (both sampled over $1,000$ random node (pairs)) in Table 4.1 and Table 4.2. In the following columns, we list the exact radius R, diameter D, center size $|C|$ and periphery size $|P|$ of the graph. In the last four columns of the table, the number of iterations I_R, I_D, I_C and I_P to compute each of these four respective measures is listed.

4.4.2 Results

As noted in the previous chapters, the proposed bounding algorithm gives a large improvement over the traditional APSP approach. For all 75 graphs, varying in size up to 60 million nodes and over 1.5 billion edges, the number of iterations is extremely low with respect to the number of nodes n. Note that small differences in the number of iterations compared to the results in Chapter 2 can sometimes be observed, which is due the fact that some minor optimizations have been done in terms of whether the algorithm starts by selecting the node with the smallest lower bound or the largest upper bound.

In the beginning of this chapter, we argued that which measure is being computed significantly influences the number of iterations of the bounding algorithm. The fact that the number of iterations differs depending on the computed measure is already shown in Table 4.2, where most of the time for each measure the number of iterations is different. Furthermore, the size (and therewith the contents) of the set of candidate nodes also differs significantly, as shown in Figure 4.1, in which the number of iterations versus the number of candidate nodes at that iteration is shown for each of the four measures.

4.4.3 Correlation with graph properties

To get an idea of the link between different graph measures and the performance of the bounding algorithm, Table 4.3 gives an overview of the Pearson correlation coefficient of the different graph properties and the number of iterations to compute

Dataset	n	m	$\overline{d}$	$\overline{e}$	R	D	$\|C\|$	$\|P\|$	I_R	I_D	I_C	I_P
Yeast [64]	1.5K	4K	7.1	13.3	11	19	48	4	7	16	65	29
Petster-Hamster [74]	1.8K	25K	3.5	9.7	7	14	2	3	3	3	4	3
Cora [122]	2.5K	10K	6.2	13.3	10	19	1	2	3	6	51	21
Mus-Musculus [82]	3.7K	10K	7.1	13.5	10	20	2	2	5	5	32	13
PPI-Dip-Swiss [82]	3.8K	24K	4.1	8.1	6	12	2	4	3	3	63	8
CA-GrQc [87]	4.2K	27K	5.8	11.6	9	17	13	8	9	24	22	22
P2P-Gnutella08 [86]	6.3K	42K	4.6	7.2	6	9	856	55	21	473	1,758	1,931
Wiki-Vote [85]	7.1K	201K	3.2	5.5	4	7	121	46	2	9	2,638	173
P2P-Gnutella09 [86]	8.1K	52K	4.8	7.5	6	10	129	9	7	179	308	1,235
CA-HepTh [87]	8.6K	50K	5.8	12.6	10	18	74	4	6	14	78	33
P2P-Gnutella06 [86]	8.7K	63K	4.6	7.4	6	10	338	4	19	30	1,675	864
P2P-Gnutella05 [86]	8.8K	64K	4.6	7.1	6	9	824	39	12	837	2,262	2,594
PGPgiantcompo [14]	11K	49K	7.8	16.5	12	24	2	3	2	2	20	3
P2P-Gnutella04 [86]	11K	80K	4.7	7.4	6	10	214	12	8	309	843	1,522
CA-HepPh [87]	11K	235K	4.6	9.4	7	13	12	17	9	20	38	62
GoogleNw [108]	16K	297K	2.5	4.7	4	7	5,267	93	2	2	5,700	36
CA-AstroPh [87]	18K	394K	4.2	10.0	8	14	139	12	11	18	244	102
Dip20090126 [121]	20K	82K	8.2	21.9	15	30	1	2	5	5	6	5
CA-CondMat [86]	21K	183K	5.5	10.7	8	15	6	11	3	13	255	53
Cond-mat-95-99 [13]	22K	117K	6.1	9.2	8	12	1,306	4	181	498	3,637	1,896
P2P-Gnutella25 [86]	23K	109K	5.6	8.6	7	11	983	5	12	862	1,367	2,857
P2P-Gnutella24 [86]	26K	131K	5.4	8.3	6	11	1	9	4	8	254	328
Cit-HepTh [86]	27K	704K	4.5	10.2	8	15	4	4	3	5	277	31
Email-Enron [73]	34K	362K	4.0	8.7	7	13	248	11	3	10	304	21
Cit-HepPh [86]	34K	842K	4.4	9.2	7	14	1	2	9	9	18	28
P2P-Gnutella30 [86]	37K	177K	5.6	8.7	7	11	602	29	20	1,526	2,234	5,409
PPI-gcc [82]	37K	271K	7.2	18.3	14	27	8	4	5	6	12	29
Brightkite [32]	57K	426K	5.0	11.8	9	18	1	4	2	2	263	3
P2P-Gnutella31 [87]	63K	296K	6.1	9.0	7	11	55	118	12	4,865	1,017	13,167
Soc-Epinions1 [114]	76K	811K	4.3	9.9	8	15	614	6	2	8	6,566	20
Soc-Slashdot [88]	82K	1.0M	4.0	8.9	7	13	484	3	3	5	778	36
Wave [137]	156K	2.1M	23.8	41.8	31	56	17	3	18	65	104	118
Itdk0304 [121]	191K	1.2M	6.8	17.1	14	26	155	7	6	19	619	42
Gowalla-edges [32]	197K	1.9M	4.4	10.8	8	16	1	29	5	5	7	15
M14b [137]	215K	3.4M	23.3	39.2	26	51	1	45	37	51	60	244
Citeseer [20]	221K	1.0M	8.4	32.0	26	52	2	3	3	3	31	3
Email-EuAll [87]	225K	680K	4.0	10.0	7	14	1	48	3	3	7	3
Web-Stanford [88]	255K	3.9M	6.1	107	82	164	1	3	5	5	6	6

Table 4.1: Results of the BoundingExtrema algorithm applied to 75 large graphs.

Dataset	n	m	$\overline{d}$	$\overline{e}$	R	D	$\|C\|$	$\|P\|$	I_R	I_D	I_C	I_P
Amazon0302 [83]	262K	1.8M	8.5	24.7	19	38	1	7	5	5	5	5
Com-dblp [144]	317K	2.1M	6.7	15.3	12	23	3	4	3	8	358	43
Cnr-2000 [19]	326K	5.5M	10.5	23.7	17	34	2	3	7	7	8	7
Web-NotreD [12]	326K	2.2M	7.4	27.9	23	46	12	172	3	3	38	3
MathSciNet [116]	333K	1.6M	7.6	16.1	13	24	317	9	16	20	731	53
Com-amazon [83]	335K	1.9M	11.6	32.8	24	47	21	5	3	7	52	14
Amazon0312 [83]	401K	4.7M	6.5	13.8	11	20	194	14	20	20	587	77
Amazon0601 [83]	403K	4.9M	6.6	16.7	13	25	69	25	3	28	189	21
Amazon0505 [83]	410K	4.9M	6.5	14.8	11	22	1	11	3	3	209	9
Auto [137]	449K	6.6M	37.9	62.9	47	82	40	4	50	276	311	338
Dblp [121]	511K	3.7M	6.2	14.8	12	22	72	9	7	59	191	82
Ydata-ysm [82]	653K	4.6M	5.9	15.7	12	24	4	3	5	5	8	5
Web-BerkStan [88]	655K	13M	7.1	126	104	208	1	2	5	5	6	5
Web-Google [88]	856K	8.6M	6.0	15.4	12	24	1	11	5	5	196	12
EU-2005 [121]	863K	32M	4.9	14.0	11	21	3	4	5	16	7	232
Imdb [82]	880K	75M	4.1	9.4	8	14	19,751	24	8	32	20,797	276
RoadNet-PA [88]	1.09M	3.1M	325	617	402	794	2	4	11	61	16	82
Youtube [103]	1.13M	6M	5.4	14.9	12	24	2	11	2	2	646	2
RoadNet-TX [88]	1.35M	3.8M	424	802	540	1,064	3	11	13	78	15	113
IN-2004 [19]	1.35M	26M	8.7	26.5	22	43	49	11	3	14	441	34
Flickr [103]	1.62M	31M	5.2	15.0	12	24	17	3	3	3	92	4
Soc-pokec [123]	1.63M	45M	4.7	9.2	7	14	2	3	3	3	8,724	21
AS-Skitter [86]	1.69M	22M	4.8	21.2	16	31	5	2	4	6	9	7
RoadNet-CA [88]	1.96M	5.5M	329	664	494	865	2	4	29	178	30	211
Enwiki-20071018	2.07M	85M	3.2	6.0	5	9	16,277	6	4	7	19,109	540
Wiki-Talk [88]	2.39M	9.3M	3.9	7.5	6	11	2,385	2	3	7	12,583	157
Orkut [103]	3.07M	234M	4.2	7.1	5	10	2	2	13	130	1,063	212
Cit-Patents [86]	3.76M	33M	8.0	17.8	14	26	4	4	13	95	50	167
LiveJournal1 [144]	4.00M	69M	5.4	13.6	11	21	31	6	3	8	2,923	19
LiveJournal2 [88]	4.84M	86M	5.5	12.8	10	20	1	2	3	6	2,103	12
LiveJournal3 [103]	5.19M	97M	5.2	15.3	12	23	18	5	3	5	89	17
P2P [82]	5.38M	284M	3.8	6.7	5	9	762	27	12	58	1,115	4,015
Hyves [128]	8.08M	912M	4.8	16.1	13	25	410	11	3	7	8,612	21
Arabic-2005 [19]	22.6M	1.10B	7.3	29.6	24	47	3	7	3	9	46	49
Wikipedia-en [74]	25.9M	1.20B	3.7	50.0	43	85	6	3	3	4	8	5
Web [82]	39.3M	1.56B	7.1	19.9	17	32	10,606	50	95	78	10,673	150
Facebook [116]	58.8M	184M	7.9	15.4	13	24	3,198	9	17	21	5,204	125

Table 4.2: Continuation of Table 4.1 (K = thousand, M = million, B = billion).

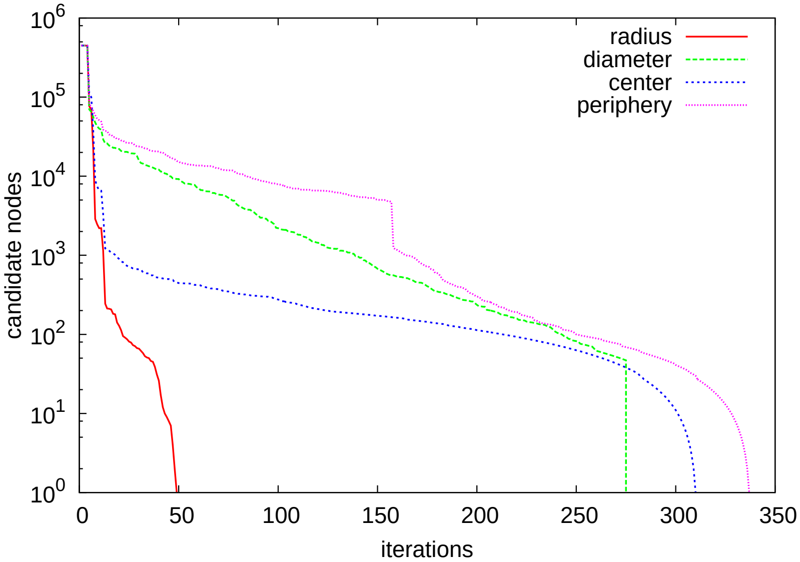

Figure 4.1: Number of candidate nodes (vertical axis) vs. number of iterations (horizontal axis) for computing each of the four measures for the AUTO dataset.

each of the measures. Here, a value close to zero indicates no significant correlation, whereas the correlation is higher as the value of the coefficient approaches 1 or -1. We will say that a numeric value greater than 0.4 (or smaller than -0.4) indicates that there is some correlation between the two variables (values in bold), whereas a value greater than 0.8 (or smaller than -0.8) indicates a definite correlation (value in bold and italics).

In Chapter 3 we have already noticed that when the difference between the radius and diameter is very large (relative to the radius and/or diameter itself), then the number of iterations is typically very low, reflected in the last row of Table 4.3 by the correlation of the number of iterations for computing the radius, diameter and periphery with $(D(G) - R(G))/D(G)$. So, when the eccentricity distribution is less wide, it is harder to compute the various measures. Table 4.3 furthermore shows that the number of iterations to find the center I_C is correlated with the center size $|C(G)|$ itself. A related observation was made in [99], where it was shown that the center size can only be determined using a number of iterations which is greater than the size of the center, as the eccentricity of each of these nodes has to be computed explicitly to confirm whether or not it has an eccentricity value equal to the radius. We do note that the proposed node selection strategy is apparently able to *identify* center candidates very efficiently, as the difference between the center size and the

Measure	I_R	I_D	I_C	I_P
Nodes n	0.188	-0.061	0.221	-0.058
Links m	0.205	-0.060	0.244	-0.033
Average degree m/n	-0.034	-0.117	**0.481**	0.014
Average distance $\overline{d}$	-0.059	-0.040	-0.059	-0.054
Density $m/(n(n-1))$	-0.074	-0.048	-0.094	-0.059
Average eccentricity $\overline{e}$	0.069	-0.030	-0.110	-0.075
Radius $R(G)$	0.059	-0.028	-0.118	-0.074
Diameter $D(G)$	0.051	-0.033	-0.119	-0.078
Center size $\|C(G)\|$	0.155	-0.030	***0.870***	-0.010
Periphery size $\|P(G)\|$	0.039	**0.445**	0.069	**0.424**
$(D(G) - R(G))/D(G)$	**-0.418**	**-0.509**	-0.173	**-0.541**

Table 4.3: Correlation of different graph measures with the number of iterations of the bounding algorithm as displayed in Table 4.1 and Table 4.2.

number of iterations is not that large, but the different between n and the number of iterations is. The number of iterations to compute the center also appears to be correlated with the average degree.

The above observations regarding the number of iterations to compute the center do not hold for the eccentricity value of periphery nodes, which can also be derived using the proposed bounds, as we already showed during the example run in Section 2.4.5. The diameter and periphery appear to be dependent on the size of the periphery. This can be explained by the fact that if the periphery is large, then the number of nodes with an eccentricity value equal to the diameter minus one is probably also large, meaning that it may be hard for the algorithm to select correct diameter-realizing nodes by taking a node with the largest upper bound, as there may be many of such nodes.

Clearly, for each of the measures the performance does not appear to be dependent of the number of nodes, links, the average distance, density, average eccentricity or the radius or diameter itself, demonstrating the scalability of the algorithm. Finally we mention that the variables investigated in Table 4.3 are not independent. Specifically, we note that a high average degree means that there are probably many nodes that belong to the center, explaining the correlation of I_C with both the center size and the average degree. This dependency is further investigated in the following section.

4.5 Experiment on a synthetic graph

In Section 4.4.2, we demonstrated how the number of iterations to compute the center appears to correlate with the average degree and the center size itself. To verify these correlations based on more confined experiments, we generate a number of synthetic graphs with increasing average degree.

The synthetic graph is generated as follows. First, a graph $G = (V, E)$ is initialized with $|V| = 1000$ nodes $E = \emptyset$ (so, a graph with no edges). Then For N iterations, an undirected edge (u, v) is added to E, where u is selected at random, and v is selected with a probability proportional to its degree, reflecting the well-known preferential attachment model [104]. This results in a graph in which the average degree grows gradually, as the number of nodes in the largest component will quickly be consistent and the number of edges grows linearly.

We performed the procedure explained above for $N = 75,000$, resulting in a graph with $150,000$ links and an average degree of 150. After $4,222$ iterations, 99% of the nodes resided in the giant component, and after $6,399$ iterations this value increased to 100%. At each iteration, after an edge is added, the center is re-computed using the proposed algorithm. The results are depicted in Figure 4.2. First of all, we observe that indeed the center size and the number of iterations are correlated, as they exhibit a

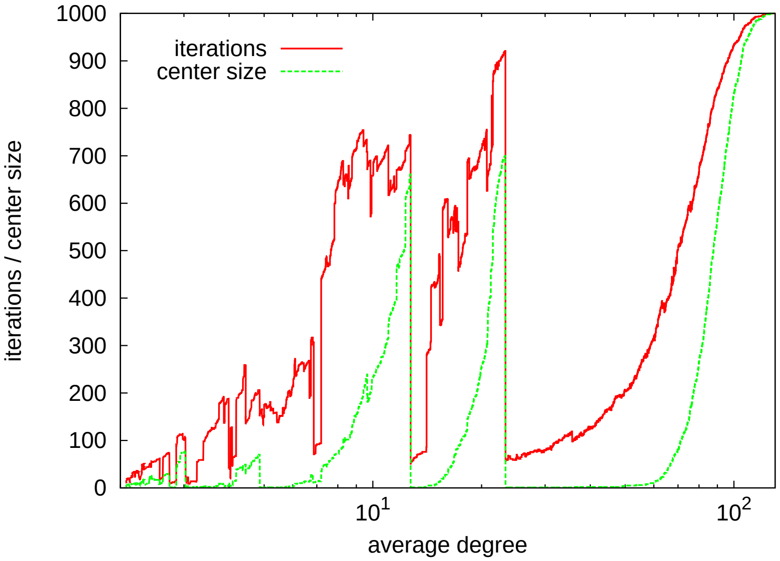

Figure 4.2: Center size and number of iterations to compute the center (vertical axis) vs. average degree (horizontal axis) for a synthetic graph.

similar curve. The various peeks around various low average degrees and degrees of 4, 13, 24 indicate a drop in the radius, resulting in a smaller center (size). This explains why the correlation between the average degree and the number of iterations is not high, but certainly notable.

4.6 Conclusion

In this chapter, the results of applying the bounding framework for computing extreme graph measures (radius, diameter, center and periphery) to a large set of 75 real-world graphs are discussed. We have identified various factors that influence the performance by looking at the correlation of the performance with different graph properties, and found the following correlations:

- The radius, diameter and periphery are harder to compute using the bounding framework when the eccentricity distribution width is relatively small ($(D(G)-R(G))/D(G) < 0.5$).
- The diameter and periphery become harder to compute using the bounding framework as the size of the periphery increases.
- The center is harder to compute using the bounding framework when the center itself is very large or when the average degree is very high (two variables which we believe to be dependent).

Regardless of the correlations presented above, the proposed bounding technique always improves significantly upon the traditional APSP-based method for computing these measures.

5

Adaptive Landmark Selection Strategies for Fast Shortest Path Computation

This chapter considers the task of answering shortest path queries in large real-world graphs. The traditional Breadth First Search (BFS) approach for solving this problem is too time-consuming when networks with millions of nodes and possibly billions of edges are considered. A common technique to address this issue uses a small set of landmark nodes from which the distance to all other nodes is precomputed in order to then answer arbitrary distance queries by navigating via one of the selected landmarks. The problem of finding an optimal set of landmarks has been shown to be NP-hard. This chapter investigates the graph characteristics that determine the successfulness of a landmark selection strategy. Then, we propose a new adaptive heuristic for selecting landmarks that does not only pick central nodes, but also ensures that these landmarks properly cover different areas of the graph. Experiments on a diverse set of large graphs show that the proposed selection strategy and assisting node processing technique can efficiently estimate the node-to-node distance in graphs with millions of nodes with very high accuracy, while using the same amount of precomputation time as previously proposed strategies. This chapter is based on:

- F. W. Takes and W. A. Kosters. Adaptive landmark selection strategies for fast shortest path computation in large real-world graphs. In *Proceedings of the IEEE/ACM International Conference on Web Intelligence (WI 2014)*, pages 27–34, 2014

5.1 Introduction

A large part of computer science research deals with finding or computing simple paths, shortest paths or distances between objects in a dataset that can be modeled as a graph. An example of a path-related query is a request for a simple yes or no answer to the question of whether or not two nodes are connected, solving the so-called *reachability* problem [145] in graphs, with well-known applications in for example XML parsing and ontology querying [55]. In transportation science, the focus of solvers for so-called vehicle routing problems (VRPs, see [126]) is to service a set of nodes (customers) by *finding paths* (routes) that are as short as possible An optimal route is not necessarily required, and when large graphs are considered, not even computable in polynomial time [81]. In this chapter we will focus on the problem of finding the *length of a shortest path* between a given pair of nodes, assuming that all nodes in the graph are connected. More specifically, we consider the task of answering *distance queries* in large graphs, which has been shown to be a complex and challenging task [121]. By *large graphs* we mean that the distance matrix cannot be stored in main memory, and algorithms with quadratic space or time complexity in the number of nodes or edges are not acceptable. Common examples of large graphs include social networks, biological networks, communication networks and webgraphs.

The problem of efficiently computing the shortest path length, i.e., the *distance* between two nodes in a graph, has been extensively studied [150]. In unweighted graphs, which we mainly consider in this chapter, finding a shortest path originating from a particular node can be done by performing a Breadth First Search (BFS) until the goal node is found. For a graph with n nodes and m edges, one BFS considers each edge at most once, realizing a time complexity of $O(m)$. Doing this for each of the n nodes in the graph results in a complexity of $O(mn)$ for determining the shortest path length between all possible pairs of nodes, essentially solving the well-known All Pairs Shortest Path (APSP) problem. Clearly, for the large graphs that are nowadays studied a traditional brute-force approach is not feasible in terms of time and memory consumption.

Because of these complexity issues, approximation and estimation techniques have been introduced, most notably methods based on so-called *beacons* [72] or *landmarks* [112] that do some precomputation in order to then answer a shortest path query very quickly. Often, a small set of landmarks (nodes) is selected, for which the actual distance to every node is precomputed. When a distance query is received, an approximation based on the distance to and from these landmarks is given, using some smart lower and upper bounds on the landmark distances, assisted by checks for trivial cases in which an exact answer can be returned. Typically, the precomputation step is orders of magnitude faster than computing all the shortest paths, the size of the landmark set is orders of magnitude smaller than the number of nodes n, and the distance query time is orders of magnitude faster than the traditional BFS query time. Ultimately, there is a trade-off between space, precomputation time, query time and accuracy [121].

Although a *random* set of landmark nodes for estimating the shortest path lengths already works quite well [72], it has been shown that a careful selection strategy, for example based on nodes with a high centrality value [112], or based on a tree which covers different areas of the graph [5, 65] can greatly improve the performance of the landmark method. Clearly, not every selection strategy performs well on every type of graph, and choosing the correct landmark selection strategy can be of great influence on the accuracy of the method for a particular graph. The problem of selecting the *perfect* minimal set of landmarks, has been proven to be NP-hard [112].

The main contribution of this chapter consists of a careful analysis of what makes a certain landmark selection strategy perform well. We will look at the nodes for which the error is high, and attempt to characterize these nodes by their position in the graph. Based on the obtained insights, we propose two new techniques that attempt to improve landmark selection techniques based on common centrality measures.

The rest of this chapter is structured as follows. In Section 5.2 we consider some notation and precisely formulate the problem statement and landmark approach. Next, related work is discussed in Section 5.3, after which we explain the landmark framework in detail in Section 5.4. Most notably, two new landmark strategies are proposed in Section 5.5. In Section 5.6 we perform a number of experiments to compare the suggested techniques on a set of large real-world graphs. Finally Section 5.7 summarizes the chapter and provides suggestions for future work.

5.2 Preliminaries

In this section, basic notation is briefly discussed, a formal problem statement is given, and finally the landmark approach and its constraints are explained.

5.2.1 Notation

We consider an undirected and unweighted graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. Because the graph is undirected, each edge is included twice, so $(u, v) \in E$ iff $(v, u) \in E$. Then, a path is defined as a sequence of nodes connected by edges. A shortest path between two nodes $u, v \in V$ is a path consisting of a minimal number of edges that connects the two nodes. The length of this shortest path, or the *distance* $d(u, v)$ between nodes u and v, is simply the number of edges in such a shortest path. The assumption is that G is connected, i.e., $d(u, v)$ is finite for all nodes u and v. The *degree* $deg(v)$ of a node v is the number of edges connected to that node. We assume that the graphs are sparse, meaning that m is much smaller than the maximum number of edges $n(n-1)$.

5.2.2 Problem definition

We consider the problem of accurately and efficiently *estimating* $d(u,v)$ for any given pair of nodes $u, v \in V$. By *accurate*, we mean that the estimated value should not differ too much from the actual distance value, i.e., the error, which we will define more precisely in Section 5.6.2, has to be as low as possible. By *efficient*, we mean that the computational step of one distance estimation should be significantly faster than one simple BFS, which can be done in $O(m)$. Practically speaking, it should be possible for large graphs to estimate thousands of these distance values in a matter of seconds. The computational step of a distance estimation, which we call the *query time*, should only iterate over the set of nodes (or a subset), meaning that it should be done in at most $O(n)$ time.

To realize a low shortest path computation time, a relatively short *precomputation phase* is allowed. In the precomputation phase, an algorithm typically iterates over the set of nodes and/or edges a constant number of times, for example to perform a few real BFS runs (each taking $O(m)$ time). So for a reasonably small integer constant $c > 0$, the *precomputation time* should be restricted to cm. The same requirement holds for the space complexity: the precomputation data should take no more memory than the graph data itself.

5.2.3 Landmarks

As a precomputation step, we select a set of landmarks $B \subseteq V$ consisting of $k = |B|$ nodes (with $k \ll n$) for which we precompute for all pairs v, w (with $v \in B$ and $w \in V$) the exact value of $d(v,w)$. Because we deal with undirected graphs, we automatically also compute $d(w,v)$. Note that k is typically very small compared to n, and thus storing $k \times n$ distances is possible. In contrast, storing $n \times n$ distances, i.e., the full distance matrix, is not possible. Indeed, for $k = n$ we would essentially be solving the All Pairs Shortest Path (APSP) problem which is prohibited due to time and memory constraints. The problem of selecting a good set of landmarks B from the original set of nodes V is considered in Section 5.4 and Section 5.5.

When we answer a distance query, i.e., a request for finding the distance $d(u,v)$ between two nodes u and v, we first check for the applicability of some trivial cases (assuming the graph is stored using adjacency lists) for which an exact distance can easily be derived:

- If u or v is a landmark, then we can return the exact value of $d(u, v)$ as this value is stored for the landmark.
- If u and v are identical, then obviously $d(u, v) = 0$.
- If u and v are direct neighbors, which can be determined in $O(\log(m/n))$ by searching for v in the sorted list of neighbors of u (or vice versa), then obviously $d(u, v) = 1$.
- If u and v are at distance 2, then we can also detect this efficiently in $O(m/n)$ as we can iterate over the sorted lists of neighboring nodes of both u and v, in search for a duplicate entry, resulting in $d(u, v) = 2$.

Any distance larger than 2 will have to be estimated using the landmark set by considering each of the k nodes in the precomputed set of landmarks. As observed in [112], due to the triangle equality, the following statement holds regarding the value of $d(u, v)$ given a set of landmarks B: $\max_{w \in B}(|d(u, w) - d(w, v)|) \leq d(u, v) \leq \min_{w \in B}(d(u, w) + d(w, v))$. Thus, by considering the precomputed distances for the landmarks, we can obtain a lower and upper bound L and U on the distance $d(u, v)$ between u and v. Here, L is at least equal to 3, as otherwise we would have found the distance as one of the trivial cases. So we have:

$$L = \max\big(\max_{w \in B}(|d(u, w) - d(w, v)|), 3\big)$$

$$U = \min_{w \in B}(d(u, w) + d(w, v))$$

When asked for an estimate, we can return L or U itself, the mean, geometric mean, or some other variation using the two variables. It turns out that using U as an estimate gives the lowest error rate [112].

5.3 Related work

The problem of exactly determining the distance between any or all pairs of nodes has been widely addressed. Initially, algorithms that do fast matrix multiplication [3] were frequently used. Such algorithms improve upon the straightforward Floyd-Warshall algorithm for solving the APSP problem, but suffer from large constants and obvious memory constraints. Other exact approaches are based on A* [51], but still have poor worst-case complexity.

Considering estimation and approximation techniques, data structures for answering distance queries were introduced under the name “distance oracles”, providing some theoretical results on the accuracy of the estimation [134]. Although elegant in design, these techniques are not very useful when graphs with many low distance values are considered [72], as the actual difference between the approximated and real distance can be large, which is undesirable in large graphs with relatively low pairwise distances. This happens to be the case in many of the real-world graphs that are nowadays studied, as they usually belong to the class of so-called small-world networks [71] with very low average pairwise distances.

Methods based on beacons [72] or landmarks [54, 112] were suggested as a better way of handling distance queries in this type of graphs. Selecting a minimal set of landmarks such that the graph is covered, meaning that the estimate for $d(u, v)$ is correct for all pairs $u, v \in V$, was shown to be NP-hard [112]. Selecting an efficient set of landmarks based on the centrality of a node or based on a “highway” [65] or a tree decomposition [5, 47] of the graph were suggested as heuristics for selecting an optimal set of landmarks. Various optimizations based on pruning the BFS, bitwise tricks and parallelism were introduced in [4]. Furthermore, the landmark selection method in evolving graphs with edge additions and deletions has been described in [136].

The landmark method has clearly been widely addressed, but because of the NP-hardness of the landmark selection problem, it remains a challenging method worth studying.

5.4 Landmark framework

This section describes the landmark framework, which consists of two parts: landmark selection and landmark processing. Landmark selection, considered in Section 5.4.1, deals with the problem of sorting the nodes based on their likeliness of being a good landmark. Section 5.4.2 is about landmark processing, which deals with the question of how and which of the identified landmarks should finally be used. Some smaller optimizations are discussed in Section 5.4.3.

5.4.1 Landmark selection

Landmark selection deals with the task of selecting a total of k nodes from the full set of n nodes that are going to serve as landmarks. As an improvement over a *random* selection of k landmarks, several landmark selection strategies based on the centrality of the nodes in the graph are suggested in [112]. The idea behind this is to compute the centrality value $C(v)$ of all nodes $v \in V$, and then select the k most central nodes (with the highest value of $C(v)$) as landmarks. In this chapter we will consider the following centrality measures:

Degree centrality

$$C_d(v) = \frac{deg(v)}{n-1}$$

Closeness centrality

$$C_c(v) = \frac{1}{n-1}\sum_{w \in V} d(v,w)$$

Betweenness centrality

$$C_b(u) = \sum_{\substack{v,w \in V \\ v \neq w, u \neq v, u \neq w}} \frac{\sigma_u(v,w)}{\sigma(v,w)}$$

Here, $\sigma(u,w)$ is the number of shortest paths from u to w and $\sigma_v(u,w)$ is the number of shortest paths that run through node v [26]. This measure can be normalized to the interval $[0;1]$ by dividing it by the largest betweenness value over all nodes.

PageRank $C_{PR}(v)$, which is the value of $PR(v)$ after iteratively (usually 100 iterations is enough for convergence) and simultaneously applying:

$$PR(v) \leftarrow \frac{1-d}{n} + d\left(\sum_{w \in N'(v)} \frac{PR(w)}{deg(w)}\right)$$

for each of the nodes $v \in V$, where $PR(v)$ is initialized to $1/n$ and $N'(v)$ is the set of nodes that links to node v and the well-known random-surfer parameter d equals 0.15 [107].

Each of the four measures is normalized to the interval $[0;1]$, where a higher value indicates that the node is more central according to the considered measure. Betweenness and closeness are two centrality measures that are just as hard to compute as the distance between all nodes (they require $O(mn)$ time). Luckily both measures can be estimated by means of sampling, reducing complexity to cm where c is the number of samples. Computing the PageRank value of all the nodes also means iterating over the set of m edges a constant number of times, resulting in a similar time complexity.

Although numerous other centrality measures have been suggested in literature, we believe that these four measures are the most common, but more importantly are of four different types. Respectively, they are based on a local property of the nodes (degree), the number of shortest paths that runs through a node (betweenness), the average distance from the node to every other node (closeness) and the centrality of the node based on a propagation model (PageRank).

Intuitively, the best nodes to be selected as landmarks for finding shortest paths length, would be the nodes with the highest *betweenness centrality* value, as the value of this measure inherently suggests that the node is part of a large portion of all the shortest paths. However, if we consider an error measure which takes the difference between the estimated and real distance into account (see Section 5.6.2), then nodes on almost-shortest paths (e.g., realizing distance plus one) are also good, but do not necessarily have the highest betweenness value. This suggests that there might be a better landmark selection strategy than simply selecting the nodes with the highest (betweenness) centrality value.

5.4.2 Landmark processing

When a set of nodes has been generated based on some (centrality) measure or strategy, a sorted list of nodes can be generated with the most central nodes on top. The simplest form of node selection is then to take the top k nodes (recall that k is the number of landmarks) without any further evaluation of how these nodes are positioned in the graph. In [112], a number of improvements over this processing technique are suggested. First, one could choose to select as landmarks the highest ranked nodes from the list from each *partition* in the graph as defined by some partitioning or clustering algorithm. Although intuitively useful, this suggested improvement did not produce a significantly much lower error, but comes with an additional computation cost and is thus not considered further in this chapter.

A second suggested processing technique is to process the list from top to bottom, but skipping nodes that are at most at distance x from previously selected landmarks. The idea behind this is that a central node that is close to previously selected landmarks does not contribute equally compared to a central node further away from previously selected landmarks. The latter optimization hints towards a second pitfall in simply using the most central nodes as landmarks, namely that central nodes are often direct neighbors. Although it turned out that $x = 1$ performed best, sometimes this processing step gives no improvement or even increases the error.

5.4.3 Optimizations

Several optimizations that can be applied after a path based on landmarks has been derived, have been suggested in [54]. Most notably, if for determining $d(u, v)$ the concatenation of paths from node u to landmark w to node v includes the same node more than once, then the intermediary nodes can be skipped, known as *cycle elimination*. Furthermore it is suggested that when two paths have been concatenated, a quick check for a *shortcut* can be done by determining for each node in the path whether its neighborhood contains any of the successors in the path, and if so, using this edge instead of the subpath to that successor.

We introduce an additional seemingly obvious optimization specifically for increasing the bound value for nodes with degree 1. If node u has a degree of 1, then u always needs its direct neighbor to navigate to every other node in the graph. Thus, for this node, the lower or upper bound of the neighboring node plus 1 can be returned. Real-world graphs typically have a power law degree distribution, and nodes with degree 1 are expected to be very common.

5.4.4 Example

To get an idea of how the landmark framework as defined in Section 5.2.3 assisted by the various discussed optimizations in Section 5.4.3 can answer distance queries, we will give various examples of such queries using the example graph shown in Figure 5.1. Note that this graph has two landmark nodes F and P.

- $d(L, J)$: Node L and J are direct neighbors ($(J, L) \in E$), so the distance is equal to 1.
- $d(B, D)$: Node B and D have common neighbor C, so the distance is equal to 2.
- $d(F, Q)$: Node F is a landmark for which all distances have been precomputed, so we can simply look up $d(F, Q)$, which is 4.
- $d(A, K)$: Using landmark node F, we find the upper bound $d(A, F) + d(F, K) \leq 2 + 2 = 4$. No further optimizations can be applied, so the (in this case correctly) assessed value is 4.
- $d(E, N)$: Using landmark P we find upper bound $d(E, P) + d(P, N) \leq 4 + 1 = 5$.

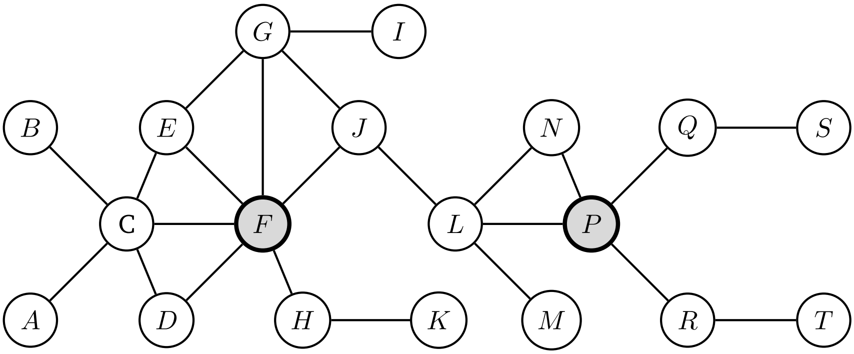

Figure 5.1: Example graph with landmark nodes F and P.

Via landmark F we find $d(E,F)+d(F,N) \leq 1+3 = 4$. The minimal upper bound and thus the (in this case correctly) assessed path length is 4.

- $d(G,M)$: Using landmark node F, we find upper bound $d(G,F)+d(F,M) \leq 4$. Alternatively, via landmark P we would find the path (G,J,L,P,L,M) with length 5. Optimizing the first path (G,F,J,L,M) via F we find that $(J,G) \in E$, meaning that we can take a shortcut by eliminating node F, resulting in path (G,J,L,M) with length 3. Note that after cycle elimination, the path via landmark P would also result in the shortest path (G,J,L,M).
- $d(I,L)$: Using landmark node F, we find upper bound $d(I,F)+d(F,L) \leq 4$. After looking for shortcuts, the intermediary node F can be removed resulting in an upper bound of 3. However, node I has a degree of 1, meaning that we could have just looked at node G and found that G and L have common neighbor J, resulting in a distance of $2+1=3$.

5.5 Balancing centrality and covering

The previous section has described two problems when it comes to using the most central nodes as landmarks. First, centrality measures often do not take into account almost-perfect distances and second, most importantly, central nodes are often grouped together, not properly covering different parts of the graph. In this section, improvements for overcoming these two problems are suggested for both the landmark selection and the landmark processing step discussed in the previous section.

5.5.1 Adaptive landmark selection

The landmark selection strategy that we propose in this chapter is *adaptive*, meaning that the strategy improves its set of landmarks based on the error reduction of its nodes. First, let us look at a preliminary figure of the performance of different selection strategies based on centrality measures in Figure 5.2. Clearly, the percentage of correctly assessed path lengths (which we will call the *success rate*) increases monotonically with the number of landmarks. An important observation is that centrality measures work regardless of the size of the landmark set: landmarks selected based on centrality measures are able to realize a significantly higher success rate than landmarks that were selected at random. The same was observed for the other real-world graphs that we studied, although there was no single best-performing centrality measure.

We furthermore note that not every landmark appears to evenly contribute to increasing the success rate: some landmarks (horizontal steps in Figure 5.2 from k

to $k + 1$ landmarks) contribute significantly more to the success rate than others. Ideally the nodes that realize a big increase in the success rate should be ranked higher than nodes that only marginally increase the success rate. Although obviously the increase realized by a landmark node is highly dependent on previously chosen landmarks, we do expect nodes with a great incremental contribution over previously selected landmarks to give a high contribution to the performance in general. This observation is the basis for the *adaptive landmark selection strategy*:

1. Sort the set of nodes V based on degree centrality, resulting in a list of ranked nodes, with the most central node v having rank $R(v) = 1$.

2. Perform the sampling phase, in which a number of BFS runs is performed, storing how many times each node v is part of one or more shortest paths.

3. Compute the value of $S(R(v))$, the success rate at each successive rank. Note that here the rank is equal to the potential landmark count. This means that we are generating the plot in Figure 5.2 for the particular centrality measure chosen in Step 1.

4. For rank $i = 1$ to n, derive for each rank i the value of $\Delta S(R(v))$: the increase in the success rate realized by the landmark node v at rank i.

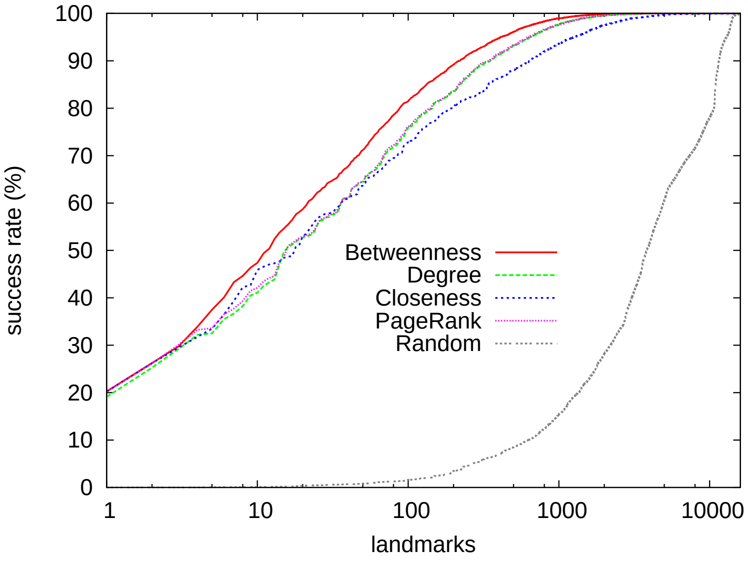

Figure 5.2: Success rate of different centrality measures as landmark selection strategies, applied to the CA-CondMat network.

5. Re-sort the list of nodes according to $\Delta S(R(v))$, resulting in a list with the (node with the) highest increase in success rate on top.

6. Use the list from step 5 as input for the processing step of the landmark framework, cf. Section 5.4.2.

In step 1, we have chosen degree centrality, because this measure does not require any additional computation time. Recall that in step 2, counting the number of shortest paths is just as complex as computing the shortest path [26]. Furthermore, performing one BFS from a particular node to every other node, and then comparing the actual distances with the estimated distances, allows us to quickly sample $n - 1$ distance computations using only one BFS. Note that the sampling procedure described above replaces the computational step of for example betweenness centrality or closeness centrality. Because the number of samples in step 2 is equal to the number of samples needed to compute the node centrality values of some centrality measure, there is no additional computation time involved in the adaptive landmark selection technique.

The intuition behind this method is that because we first sort the list based on a centrality measure and then compute the success rate difference of each node in the list, we are taking both the centrality aspect and the covering aspect into account, as nodes that do not significantly increase the success rate, apparently do not cover parts of the graph that are not already covered by other nodes. Although step 2 through step 5 could be performed in an iterative process until the error converges to a minimum, we found that one iteration always results in improvement, but more than one iteration almost never improves and sometimes even results in worse performance. The latter is likely due to the fact that after multiple iterations the influence of the centrality measure is lost. Furthermore, an iterative process requires more BFS runs and thus more computation time.

5.5.2 Greedy central neighbor processing

The second contribution of this chapter is an alternative landmark processing technique called *greedy central neighbor* processing (in short: *gcn*-processing), which works as follows. When processing the list of nodes generated using some strategy or centrality measure, we select for each node in the list h times its most central neighbor, if such a better neighbor exists and if this neighbor is not already a landmark. For example, we select for each node in the node list sorted by the success rate as a result of the adaptive landmark selection from the previous section, the neighbor with the highest degree (if that degree is higher than the degree of the currently considered node). The intuition behind this method is that it solves problems with many central nodes being clustered together, not covering the rest of the graph. Note that for each node in the list, one central neighbor at most h hops away is selected, so the landmark count k remains unchanged. Furthermore, the suggested technique is *greedy*: it does not look at the full neighborhood (which would be computationally expensive), but merely repeatedly picks the most central neighbor.

As an example, consider Figure 5.3 and Figure 5.4 that both show a visualization of the CA-HEPPH network (for a description of this graph, see Section 5.6.1). In these two visualizations of the same graph, the size of a node is proportional to its degree, and the color of a node denotes the error (see Section 5.6.2 for a description of this error measure) observed when computing a shortest path from or to that particular node. In Figure 5.3, shortest paths were computed using $k = 100$ landmarks selected using degree centrality, whereas the errors in Figure 5.4 are from $k = 100$ landmarks that were selected using random node selection, but when processing the list, each time in a greedy way selecting the most central neighbor of the considered node (with $h = 3$). All low error nodes (and thus all high degree nodes) in Figure 5.3 are grouped together in one densely connected cluster. The error is much higher for the rest of the graph, with higher errors as the distance to the high degree cluster increases. On the other hand, in Figure 5.4 the error is much lower in the entire graph, demonstrating the usefulness of the greedy central neighbor processing approach in solving the problem of all central nodes residing in one cluster.

In a way, the *gcn*-processing technique combines two centrality measures: one measure is used in the landmark selection phase and another measure is used as a guide for greedy central neighbor landmark processing. It is essentially an alternative for taking the weighted average of two measures. In order not to increase the complexity of the precomputation step, the degree can be used as a measure for determining which neighbor has to be selected. In Section 5.6 we will have a detailed look at the performance of *gcn*-processing for each of the previously discussed landmark selection strategies.

5.6 Experiments

In this section we perform experiments on a large set of networks to determine the performance of the different landmark selection techniques, specifically the two new selection and processing techniques introduced in the previous section: adaptive landmark selection and greedy central neighbor processing. We start by describing the used graph datasets in Section 5.6.1 and a verification approach in Section 5.6.2, after which we discuss the results in Section 5.6.3.

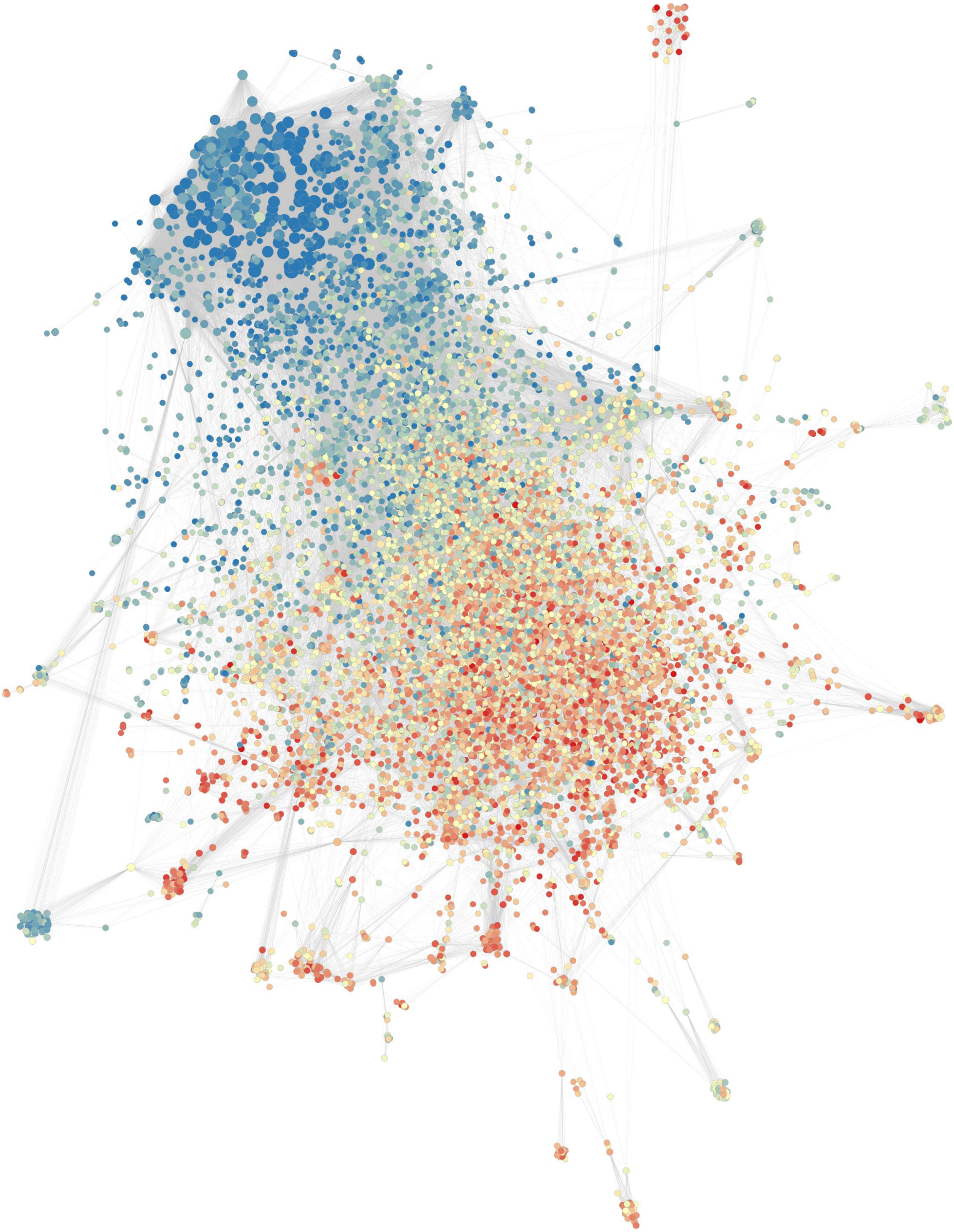

Figure 5.3: Absolute error in estimated distances in the CA-HEPPH graph from low (blue) to red (high) with 100 landmarks using *degree centrality*.

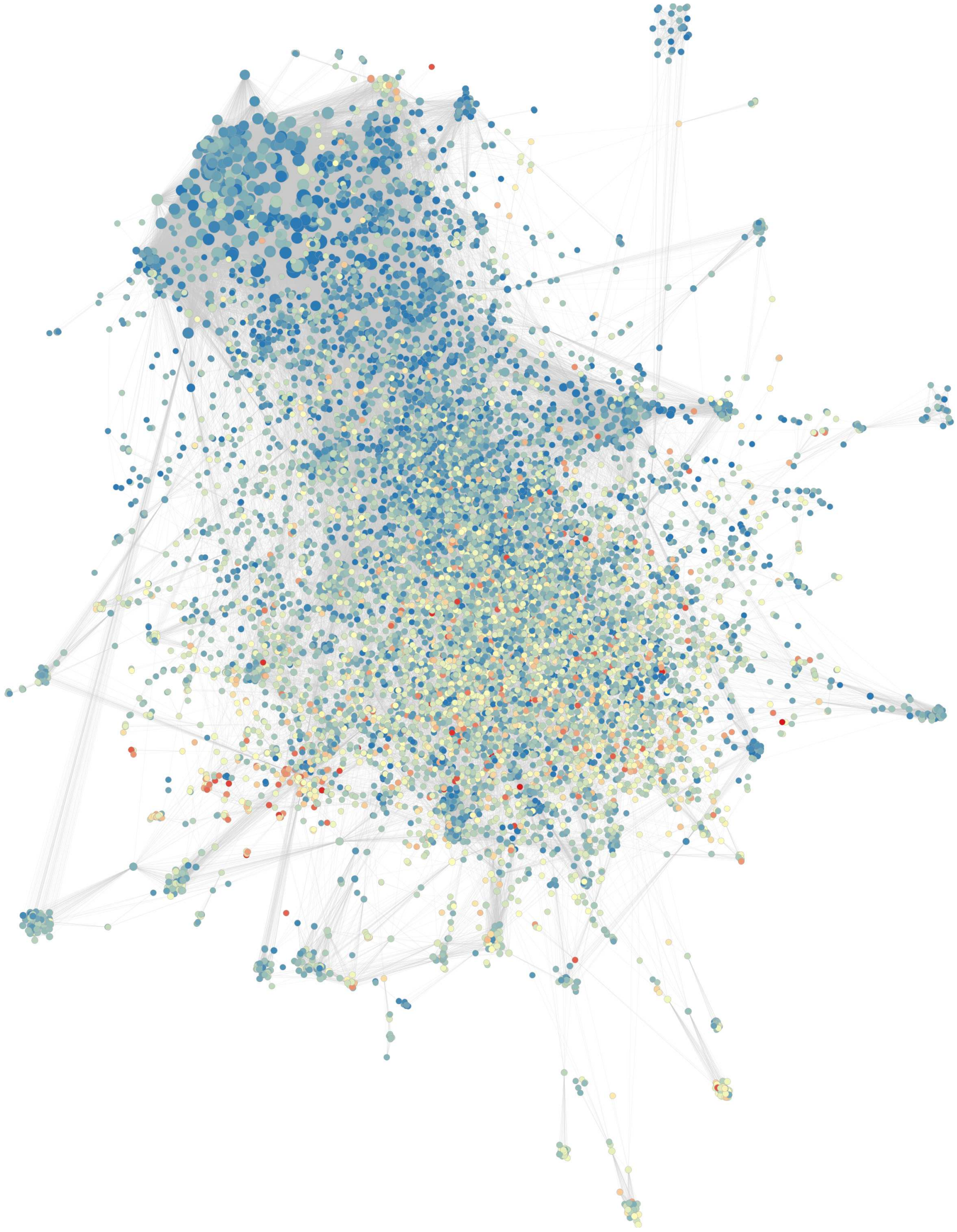

Figure 5.4: As Figure 5.3, but with random landmarks and *gcn*-processing. Visualized using ForceAtlas2 in Gephi (`http://gephi.org`).

5.6.1 Datasets

Table 5.1 gives an overview of the graph datasets used in this study, including for each graph a reference to the paper in which its properties are discussed in detail. The second column indicates the type of the graph, including (scientific) collaboration and citation networks, webgraphs, communication networks, social networks and electronic networks. All graphs represent real-world data, are typically sparse, and adhere to the small-world property [71]. For each graph, of the largest connected component of (the undirected version of) the graph, the number of nodes n, edges m and average node-to-node distance $\overline{d}$ (sampled over $1,000$ node pairs) are listed. We performed experiments for five landmark selection strategies: random selection, betweenness centrality, PageRank, degree centrality, and the newly proposed adaptive landmark selection. For each selection strategy, we experimented with six landmark processing techniques: plain top-k selection (0), skip-1 processing as described in Section 5.4.2, and greedy central neighbor processing for $h = 2$, $h = 3$, $h = 4$ and $h = 5$, as described in Section 5.5.2. Table 5.1, which is continued in Table 5.2, the column "gcn" indicates the lowest error as well as between brackets the value of h for which this error was observed.

Closeness centrality never performed better than other measures such as betweenness centrality (see for example Figure 5.2), so was left out of the result table. Furthermore, random selection with skip-1 processing and degree centrality with *gcn*-

Dataset	Type	n	m	$\overline{d}$	Random		Degree	
					0	**gcn (h)**	**0**	**gcn (h)**
CA-HepPh [86]	collab.	11.2K	235K	4.66	.509	.080 (3)	.137	.091
GoogleNw [108]	web	15.7K	297K	2.46	.224	**.000** (3)	.001	.003
CA-CondMat [86]	collab.	21.3K	182K	5.47	.551	.068 (2)	.100	.098
Cit-HepTh [86]	cit.	27.4K	704K	4.29	.562	.040 (4)	.047	.071
Enron [73]	comm.	33.7K	362K	4.05	.615	.013 (4)	.012	.102
Soc-Slashdot [88]	social	82.2K	1.09M	3.94	.764	.048 (4)	.049	.053
Dblp [144]	coll.	99.3K	1.09M	3.94	.605	.135 (4)	.113	.096
M14b [137]	elec.	100K	1.28M	52.5	1.17	.743 (3)	.270	.174
Wave [137]	elec.	156K	2.1M	22.9	.531	.461 (2)	.164	.120
Web-Stanford [88]	web	255K	3.88M	7.31	.343	.007 (2)	.007	.010
Web-Google [88]	web	856K	8.58M	6.18	.884	.149 (3)	.006	.009
Wiki-Talk [88]	comm.	2.39M	9.31M	3.91	.775	**.030** (4)	.039	.088
LiveJournal1 [144]	social	4.00M	69.3M	5.39	.831	.163 (3)	.082	.079
Hyves [128]	social	8.08M	912M	4.75	.528	.038 (3)	.034	.060

Table 5.1: Performance (error, lower is better) of different landmark selection strategies on various large real-world graphs.

processing (which is based again on degree centrality) make no sense, so these respective result columns were also left out.

5.6.2 Measurement methodology

In our experiments, we have consistently used 1% of the nodes in the connected component of the graph as landmarks, with a maximum of $k = 100$ landmarks. As this chapter specifically considers landmark selection strategies, we do not compare our methods with other distance estimation methods. For a general comparison of the landmark framework with such methods, we refer the reader to [112]. Assessing the performance of a landmark strategy can be done by computing the *error* (sampled over $1,000$ node pairs), defined as:

$$|d_{real} - d_{estimate}| \ / \ d_{real}$$

Recall that the success rate discussed in Section 5.5.1 only counted the number of times a landmark node was on a shortest path. Here, $d_{estimate}$ is the estimated distance (for the real distance d_{real}) by employing the full landmark framework as described in Section 5.4, so including checks for trivial distances and the described optimizations. Depending on the type of application that is considered, alternative error measures such as counting the percentage of distances that differ by at most 1 could be used.

The number of iterations in the precomputation step is fixed, and the final distance result is measured using the error measure. Therefore, clock time is not a relevant performance measure, nor are the specific properties of the machine used for the experiments. However, to put the results in perspective, we do mention that one BFS using our straightforward C++ code takes about six seconds for the graph consisting of 8 million nodes listed in Table 5.1. This means that the total precomputation time for approximating betweenness centrality and the adaptive landmark selection strategy, methods which both perform a total of $k = 100$ BFS runs along with some book-keeping, can be done in a few minutes, even for the largest graph. The same holds for PageRank, which can be computed in roughly the same time, also using 100 iterations. Random landmark selection and degree centrality obviously do not require any precomputation time.

5.6.3 Results and discussion

As we already demonstrated for one graph in Section 5.5, random landmark selection is clearly outperformed by centrality measures. We note that of the centrality measures, most of the time betweenness centrality has the lowest error. Degree centrality and PageRank are in most cases equally good. Because the newly introduced adaptive landmark selection technique builds upon degree centrality, we say that the new adaptive strategy is successful if it outperforms degree centrality, as otherwise the new method would not be worth the additional computation time compared to degree centrality. As Table 5.1 shows, this is the case for all 14 graphs, demonstrating the usefulness of the newly proposed adaptive selection strategy. On a number of datasets, the adaptive landmark selection strategy even outperforms betweenness centrality, a result which strengthens the claim that we made in the beginning of the chapter: nodes that are part of a large number of shortest paths do not necessarily serve as the best landmarks. The error observed when using the skip-1 optimization is diverse. Although there is a big increase in performance for the adaptive landmark selection for the Wave graph, most of the time the error is similar or worse as was also observed in [112].

The second contribution of this chapter is the greedy central neighbor processing strategy. Obviously, this method mainly assists a landmark selection strategy in the final processing phase. To get an idea of the contribution in terms of performance of *gcn*-processing, we can look at the error for a random set of landmarks as compared to a random set of landmarks processed using *gcn*-processing (so, the column of Table 5.1 titled “Random”). We see that the greedy central neighbor processing tech-

Dataset	**Betweenness**			**PageRank**			**Adaptive**		
	0	**skip-**1	**gcn (*h*)**	**0**	**skip-**1	**gcn (*h*)**	**0**	**skip-**1	**gcn (*h*)**
CA-HepPh	**.045**	.078	**.045** (2)	.140	.090	.093 (3)	.117	.103	.080 (3)
GoogleNw	.001	.004	**.000** (3)	.001	.003	.001 (2)	**.000**	.003	**.000** (2)
CA-CondMat	**.044**	.064	.045 (2)	.059	.066	.054 (3)	.064	.083	.056 (3)
Cit-HepTh	.031	.059	**.029** (3)	.048	.069	.046 (3)	.051	.086	.044 (3)
Enron	.010	.009	**.008** (2)	.011	.098	.009 (4)	.022	.145	.012 (3)
Soc-Slashdot	.081	.052	.048 (2)	.085	.046	.053 (2)	.078	.052	**.032** (4)
Dblp	**.090**	.109	.091 (2)	.103	.105	.099 (3)	.093	.102	.093 (3)
M14b	.276	.116	.267 (3)	.501	.152	.377 (3)	.059	.065	**.054** (2)
Wave	.199	.142	.194 (3)	.270	.126	.211 (3)	.121	**.079**	.096 (3)
Web-Stanford	**.003**	.006	**.003** (2)	.005	.007	.006 (2)	.010	.008	.004 (4)
Web-Google	.006	.019	**.005** (3)	.006	.006	**.005** (4)	.006	.010	**.005** (4)
Wiki-Talk	.038	.122	.038 (3)	.037	.040	.037 (4)	.036	.128	.036 (2)
LiveJournal1	**.067**	.079	.067 (2)	.075	.082	.071 (2)	.069	.074	.071 (2)
Hyves	.045	.065	.035 (3)	.035	.055	.035 (2)	.042	.069	**.029** (3)

Table 5.2: Performance (error, lower is better) of different landmark selection strategies (continuation of Table 5.1).

nique always greatly improves upon plain random landmark selection, most notably in case of the CA-HepPh and Wiki-Talk graphs, where the *gcn*-processing technique applied to a random set of nodes even outperforms degree centrality. In all other cases, random selection with *gcn*-processing does not improve upon degree centrality, suggesting that the selected nodes are merely a local minimum. We note that for $h > 4$ there was never an increase in performance compared to smaller values of h.

We also applied *gcn*-processing to the betweenness and PageRank selection methods, and there we also observe increases in performance. Apparently, although betweenness and PageRank both take the global aspect of the graph into account, locally some optimization using the *gcn*-processing technique can still be achieved. In case of the adaptive landmark strategy, the increase in performance obtained by using *gcn*-processing is diverse, but often relevant. For example in case of the Soc-Slashdot or CA-HepPh network, the error is actually significantly lower when *gcn*-processing is used.

In general we can conclude that the new landmark selection and landmark processing techniques work well for the set of real-world graphs, as shortest path lengths can be determined with an error that is consistently lower than 0.10. We note that even when the average distance in the graph is relatively high, such as for the Amazon and Wave graphs, the error remains low. Finally we note that based on the results that we obtained, the error does not appear to be influenced by variables such as the numbers of nodes or edges or the average node-to-node distance, which demonstrates the scalability of the suggested techniques.

5.7 Conclusion

The performance of the landmark methodology for assessing shortest path lengths in large real-world graphs heavily depends on the chosen landmark selection strategy. Using various experiments we have shown that the task of selecting a good set of landmarks involves at least two aspects: selecting a set of central nodes and properly covering different areas of the graph. In order to address these two aspects, in this chapter we have compared different landmark selection strategies and introduced the adaptive landmark selection strategy and the greedy central neighbor processing technique. Experimental results on a number of real-world graphs show that using the same amount of precomputation time, the proposed strategies outperform and improve previously suggested landmark selection techniques based on centrality.

The question remains whether or not it is possible to determine beforehand which landmark strategy is expected to show the best performance. In future work we would like to investigate how we can link different graph-based properties to the performance of a specific selection and processing technique in order to determine a priori a suitable landmark selection strategy. Although we have demonstrated the success of the proposed strategies on a number of real-world graphs, more research is needed to determine the worst-case performance in order to give an upper bound on the error. Furthermore we want to see if the proposed adaptive landmark selection strategy as a whole can also be applied to the problem of determining shortest path lengths in evolving graphs that are growing and shrinking as nodes and edges are added and deleted.

6

Identifying Prominent Actors in Online Social Networks using Biased Random Walks

In this chapter, the structural properties of the friendship graph of a large online social network consisting of 8 million users and close to 1 billion links are investigated. The main focus is on characterizing the prominent users that reside within the online social network based on their position in the graph. The derived structural node properties will then be used to steer an automated classification algorithm based on biased random walks for distinguishing between prominent and regular nodes (users). The effectiveness of the proposed approach is assessed using the online social network at hand, for which it is known which nodes have been manually identified as prominent individuals. It turns out that using the proposed random walk algorithm, it is possible to efficiently identify a large portion of the prominent nodes in the network, outperforming standard web-inspired measures such as HITS and PageRank. This chapter is based on:

- F. W. Takes and W. A. Kosters. Identifying prominent actors in online social networks using biased random walks. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)*, pages 215–222, 2011

6.1 Introduction

Nowadays, *online social networks* (*OSNs*) such as Facebook, Twitter and LinkedIn are extremely popular, with numbers as high as hundreds of millions of users and billions of friendship links. The main concept of these online social networks is simple: a user creates a profile with some personal attributes and then links this profile to other users, the so-called *friends*, creating a very large graph of befriended users: the *friendship graph*. In order to better understand the structure of social networks, the friendship graph is extensively being measured, modeled and mined [2, 76, 103].

In this chapter, we consider the (former) Dutch online social network HYVES, which was online for about 9 years between 2004 and 2013. We will investigate one full snapshot of the network which was obtained in September 2010. At that time, little over 8 million users participated in the friendship graph of the network, together forming well over 900 million friendship links. The main question discussed in this study is how we can *automatically* identify the most important users in this online social network, based only on the structure of the network. Being able to identify such prominent actors has various useful applications. For example, companies nowadays frequently use online social networks for their *viral marketing* [83] campaigns, in which they want to deliver a message to as many people as possible through the linking structure of the social network. Prominent nodes may just be the places where such a campaign should start in order to efficiently reach a large number of people [61].

A relevant question is then how prominence or importance of a node within a network should actually be defined for the network that we consider. While various definitions may be correct, we will assume that someone is *prominent*, or *important*, or *influential*, if he or she has some celebrity status (famous politicians, soccer players, artists, movie actors, etc.) in the real world. We believe that this definition can be justified based on the fact that both online and in the real world, celebrities have a certain status, or reputation. If a celebrity promotes a certain brand, people are far more likely to identify with that brand, compared to when a regular person would promote the brand [105]. Within the online social network Twitter, tweets originating from people like president of the United States are far more likely to be "retweeted" than when they would come from a normal person in the network. Thus, the president could be seen as a more prominent actor in the network.

In this chapter we first study the difference in characteristics between regular and prominent nodes. We consider various existing methods of determining the importance of nodes in the friendship graph of an online social network, and then introduce a new method which is based on the characteristics of the prominent nodes that we obtained. Next, we verify the performance of the discussed techniques empirically on a large complete dataset of the online social network. For this network, we know exactly which users are considered to be prominent, allowing us to verify the obtained results against a predefined ground truth.

The rest of the chapter is structured as follows. In Section 6.2 we formally define the main problem, after which we discuss related and previous work in Section 6.3. In Section 6.4 we discuss the characteristic properties of prominent nodes, and how these properties can be incorporated in a random walk algorithm. Next, in Section 6.5 we describe the dataset that we will use in Section 6.6 to compare the performance of the discussed methods. Section 6.7 concludes.

6.2 Preliminaries

In this section we will discuss some basis concepts, formulate the problem statement and describe the application domain.

6.2.1 Definitions

We are given a friendship graph $G = (V, E)$, where V is the set of $n = |V|$ nodes (individuals) and $E \subseteq V \times V$ is the set of $m = |E|$ edges (connections). The graph is undirected, meaning that the set of edges is symmetric, so if $(u, v) \in E$ then also $(v, u) \in E$. A *path* or *walk* from u to v is a sequence of edges, starting with an edge containing u and ending with an edge containing v. The distance $d(u, v)$ between two nodes u and v is defined as the length of a shortest path between these nodes. Because the graph is undirected, $d(u, v) = d(v, u)$ for all $u, v \in V$. A connected component is a maximal subset of the set of nodes V such that the any pair of nodes within this subset is connected via one or more edges.

We define the neighborhood $N(v)$ of a node v as the set of nodes at distance 1 of v, more specifically: $N(v) = \{u \in V \mid (u, v) \in E\}$. The *degree* of a node v is defined as the number of edges starting (or ending) at node v, i.e., the size of the neighborhood of that node: $deg(v) = |N(v)|$.

6.2.2 Problem statement

Amongst the nodes in the network, there is an initially unknown set $W \subseteq V$, of size $k = |W|$, which contains the nodes that are considered to be “prominent”. Logically, $k \leq n$, but in practice, k is much smaller than n, as only a small portion of the nodes is typically considered to be prominent. The main goal is to find, given *only* the graph $G = (V, E)$, an as small as possible subset $I \subseteq V$ such that $|I \cap W|$ is maximal, i.e., we are trying to find as many prominent nodes as possible.

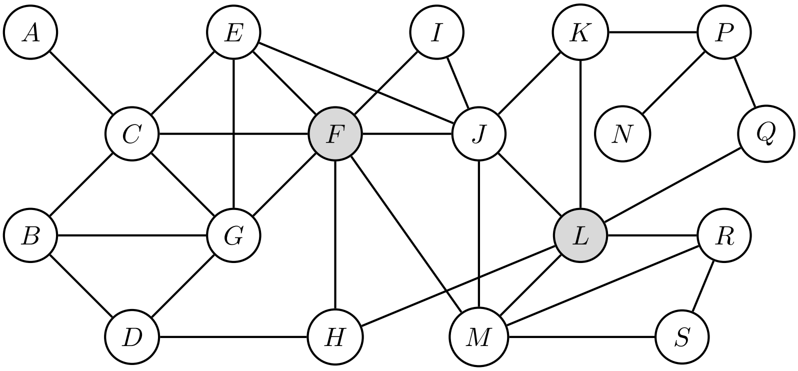

Figure 6.1: A graph consisting of 18 nodes of which 2 nodes (F and L) are manually labeled as "prominent".

In this chapter we describe various existing, derived, and new methods for determining node importance. For each of these methods M we assign a normalized value $C_M(v) \in [0, 1]$ to each node $v \in V$ which determines its importance. We will assume that higher values indicate a higher level of importance. In order to determine the performance of a method M, we sort the list of nodes by their importance value $C_M(v)$ in descending order, and define I to be the top ℓ nodes of this sorted list. The *precision* and *recall*, $|W \cap I|/|I|$ and $|W \cap I|/|W|$, respectively, will ultimately determine the performance of a method M. More generally, we can say that the F-measure, $(2 \times \textit{precision} \times \textit{recall})/(\textit{precision} + \textit{recall})$, measures the balance between the two. Note that if $\ell = k$, precision, recall and F-measure are equal.

An example of a network with 18 nodes of which 2 nodes (F and L) are manually labeled as "prominent", is given in Figure 6.1. If some method would determine that nodes F and J are the prominent nodes, then $W \cap I = \{F, L\} \cap \{F, J\} = \{F\}$ and the performance of this method (in terms of both precision and recall) would be 50%.

6.2.3 Online social networks

Topological properties of online social networks have been studied in great detail [103]. Social networks are usually sparse and contain one large connected component consisting of the majority of the nodes, called the *giant component*. Often, there are a few smaller isolated communities, as well as various singletons [2]. Furthermore, it is well-known that the structure of online social networks resembles that of real-life social networks [139]. Online social networks generally belong to the class of *small-world networks* [140]. Such networks are characterized by relatively small pairwise distances between nodes, i.e., the average distance between two nodes is very small (typically around four to eight) compared to the total number of nodes (easily more than one million). Online social networks also tend to exhibit a node degree distribution that follows a power law: there are relatively few nodes with an exceptionally high degree, and many nodes with a low(er) degree. The high degree nodes function as hubs, and are often grouped in a densely connected core, realizing the short pairwise distances between the more “peripheral” nodes.

6.3 Related work

Various studies have addressed the problem of identifying the prominent actors within large (online) (social) networks. Some related work deals with finding experts, or who can be trusted within some semantic social network [50, 148]. We will distinguish from these methods by not considering semantics, but only structural properties of the nodes.

Centrality measures have been popularized by social scientists as possible measures for the importance, or “prestige” of a person within a social network [139]. These measures identify nodes that have a central position based on the structure of the network. Such a central position usually means that the node is connected to many other nodes, possibly indirectly via some (short) path(s). *Degree centrality* is by far the simplest and most common measure, and is in case of an online social network simply equal to the number of friends of a user. As we will see later on, the number of friends is a good indication of the prominence of a node, but definitely not perfect. The complexity of computing other more complex distance-based centrality measures is in the order of $O(mn)$ or worse [27], and therefore not considered in this chapter.

Propagation-based methods such as PageRank [107] are known to be very successful in determining the importance of web pages [80], citation networks [31] and Wikipedia articles [69]. Therefore, in this study we will also consider the PageRank measure C_{PR} as defined in Section 5.4.1 as a method for identifying the prominent actors in our online social network. We will furthermore consider HITS:

Hyperlink Induced Topic Search (HITS) C_{HITS}. HITS [70] is a technique for assessing node centrality which assigns a hub score $h(v)$ and an authority score $a(v)$ to every node v in the graph, both initialized to 1. Then, for a certain number of iterations (100 iterations is usually enough for convergence), for each node v the value of $a(v)$ is set to the sum of the (normalized) $h(u)$ values of the nodes u for which there exists a link (u, v), after which for each node v the value of $h(v)$ is set to the sum of the (normalized) $a(w)$ values of the nodes w for which there exists a link (v, w). For the centrality measure C_{HITS}, we use the authority score $a(v)$.

The NodeRanking algorithm was proposed in [113] as a method based on random walks for determining the importance of authors in a directed citation network. Random walk algorithms generally traverse the graph, moving to a random neighbor with probability $1 - p$, and jumping to a random node with probability p. The distinguishing property of the NodeRanking algorithm is that it jumps with a probability depending on the degree of the current node, where a low degree indicates a high jumping probability. In Section 6.6.2 we will compare approach discussed in the next section with each of the algorithms discussed above.

6.4 Prominent nodes

We will now outline the proposed approach for finding the prominent nodes in an online social network. First we sketch the expected characteristic properties of the target nodes. After that, we will describe an algorithm based on random walks which uses these node properties to guide the walk towards the prominent nodes.

6.4.1 Node properties

The simplest intuition that we have about prominent people, is that they have a large number of connections. Therefore we expect the degree of a node to play a great role in determining the importance of a node. So we could state that degree centrality, determining the importance of a node v based on its number of connections, could be a good first indication of importance, formally:

$$C_d(v) = \frac{|N(v)|}{n-1}$$

However, there may be nodes in the graph with many connections, that are not prominent, or vice versa, prominent people with a smaller number of connections.

Let us recall several observations regarding social networks in general, which have been described in literature. People tend to use social networks for two reasons: *social searching*, and *social browsing* [78]. These two terms refer to reconfirming real-life friendships online, and browsing for completely new relationships, respectively. Another common concept is that of *triadic closure*: the vast majority of all friendships formed within a social network takes place between two people who have at least one friend in common. The probability of a link being formed has been shown to increase with the number of common acquaintances [75] as well as with the degree of a node, a phenomenon called *preferential attachment* [104]. For example, in the graph in Figure 6.1, the connection (A, B) would be more likely to appear than the connection (A, K), as A and B have node C as a common friend, and A and K have no direct common friends. The connection (A, E) would in turn be more likely than (A, B), as E already has a higher degree.

Based on the observations above, we expect that a user adding someone like the president of the United States, is not within the circle of friends of the president, making this friendship more like a result of the aforementioned social browsing instead of searching. More generally, we argue that the friends of prominent nodes have fewer connections in common than regular nodes. We call this concept the *neighborhood density* (*nd*) of a node:

$$C_{nd}(v) = 1 - \sum_{\substack{w \in N(v) \\ |N(w)|>1}} \frac{|N(w) \cap N(v)|}{(|N(w)| - 1) \cdot |N(v)|}$$

Here, the numerator defines the number of common connections, whereas the denominator normalizes the result so that it is independent of the degree of v or the degree of w. If $|N(v)| > 1$, $C_{nd}(v)$ is minimal in case $N(v)$ is fully connected, and becomes larger as a smaller fraction of the neighborhood is interconnected. The proposed measure differs from related measures such as the clustering coefficient in a sense that this measure normalizes for both the neighborhood size of the considered node as well as the neighborhood of each of the adjacent nodes.

6.4.2 BiasedRandomWalk

We believe that a combination of the two measures from Section 6.4.1 will be able to identify a large portion of the various prominent actors. Therefore we devised an algorithm based on random walks, which has a parameterized bias towards each of these properties. The random walk algorithm has as an advantage that it only needs local information to determine the next state of the algorithm, allowing the algorithm to run efficiently even when the entire graph can not be stored in main memory.

The proposed algorithm, called BIASEDRANDOMWALK (BRW), takes as input an unweighted graph $G = (V, E)$ and parameters N, p and α, and outputs a function value $C_{BRW}(v)$ for each node $v \in V$ in the graph, determining its importance. Here N is the number of steps in the random walk algorithm, p is the jumping probability (which we fix at 0.15 as suggested in literature [84]), and α is used to define the focus on either one of the two measures that we discussed.

The procedure is outlined in Algorithm 1, and works as follows. After setting some initialization values in lines 3–6, the algorithm starts by selecting a random node from V (line 7). After that, for N iterations, the algorithm repeatedly increases the function value (line 9) of the current node v by $1/N$ (to keep the final function value within $[0; 1]$). Then, the algorithm either selects a new node from the neighborhood $N(v)$ of

Algorithm 6.1 BiasedRandomWalk

```
 1: Input: Graph $G = (V, E), N, p, \alpha$
 2: Output: List $C_{BRW}$, containing $C_{BRW}[v]$ for each node $v \in V$
 3: for $v \in V$ do
 4:     $C_{BRW}[v] \leftarrow 0$
 5: end for
 6: $i \leftarrow 0$
 7: $v \leftarrow$ RandomNodeFrom($V$)
 8: while $(i < N)$ do
 9:     $C_{BRW}[v] \leftarrow C_{BRW}[v] + (1/N)$
10:     if (rand$(0, 1) > p$) then
11:         $v \leftarrow$ BiasSelectFrom($N(v), \alpha$)
12:     else
13:         $v \leftarrow$ RandomNodeFrom($V$)
14:     end if
15:     $i \leftarrow i + 1$
16: end while
17: return $C_{BRW}[\,]$
```

v using the function BiasSelectFrom() with probability $1 - p$ (line 11), or jumps to a completely random node with probability p, denoted by the RandomNodeFrom() function (line 13).

If in the function BiasSelectFrom() a random neighbor is selected, the algorithm would be a plain random walk algorithm. However, in this specific case, the function BiasSelectFrom() selects a node with a probability dependent on different function values of the prominence measures, as we expect that each of these functions tells us something about the probability of that node being prominent. This means that given current node v, the probability $P(w)$ of selecting node $w \in N(v)$ in the next step, is equal to:

$$P(w) = \frac{\alpha C_d(w) + (1 - \alpha) C_{nd}(w)}{\sum_{u \in N(v)} \big(\alpha C_d(u) + (1 - \alpha) C_{nd}(u)\big)}$$

Here, $\alpha \in [0, 1]$ defines the focus on either one of the two measures. Not surprisingly, setting the value of α to 1 resulted in roughly the same result as degree centrality. A value of 0 for α did not find any of the prominent actors, which we believe is due to the fact that even though C_{nd} is normalized, the degree plays a significant role in identifying the prominence of a node, and very low degree nodes can still get a high neighborhood density score. In an attempt to linearly tune parameter α with steps of 0.1, it turned out that any value between 0.2 and 0.8 gave consistently better results than a lower or higher value. Thus apparently, both of the discussed measures to some extent influence the final result. Therefore we fixed the parameter to 0.5 to give equal focus to both measures. Finally, N, the number of iterations, should be set to a value significantly larger than the number of nodes n. We investigate the value of N more precisely in Section 6.6.2 when we look at the convergence of the BIASEDRANDOMWALK algorithm.

6.5 Dataset

In this chapter, we consider an anonymized full snapshot of the friendship graph of the Dutch online social network HYVES from September 2010. Some statistics such as the number of nodes and edges, the average degree and the density (defined as the number of edges divided by the maximum number of edges, i.e., $m/(n(n-1))$) of this graph are given in Table 6.1.

Although the graph has almost $10,000$ connected components (see Figure 6.2 and note the logarithmic vertical axis), the vast majority of the nodes resides within the largest connected component. According to the statistics provided on the website of the social network at the time the snapshot of the network was made, the website

Full network	
Nodes	8,113,017
Links	912,120,070
Average degree	112
Density	$1.386 \cdot 10^{-5}$
Connected components	9,926
Largest component	
Nodes	8,083,964
Nodes %	99.6%
Links	912,067,984
Links %	99.99%
Density	$1.396 \cdot 10^{-5}$
Average distance	4.75
Radius	13
Diameter	25

Table 6.1: Friendship graph statistics.

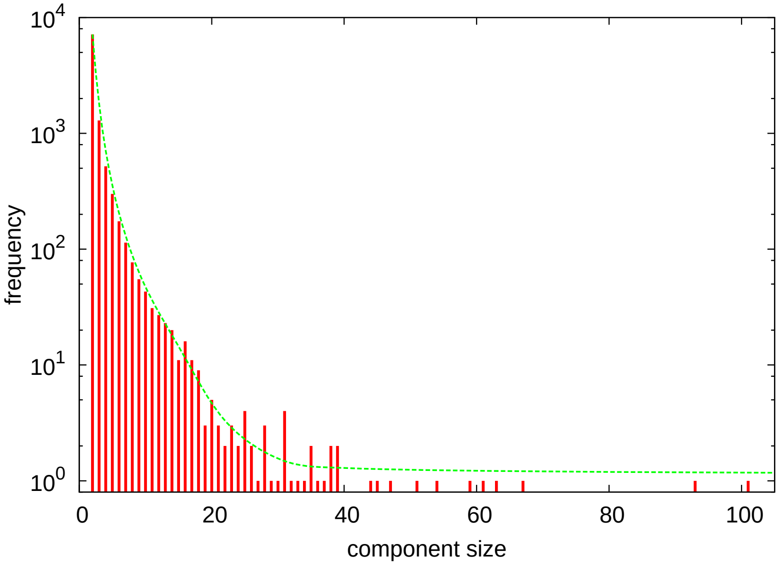

Figure 6.2: Component size distribution (excluding the largest component of $8,083,964$ nodes).

had over 11 million members. This means that there were roughly 3 million users that were not participating in the friendship graph at all. The node degree distribution of the graph is shown in Figure 6.3. This distribution follows a clear power law, and has an even longer tail than visible, going all the way up to one node with a degree of $285,827$. Note that the social network had a maximum number of friends at $1,000$, $1,500$ and $2,000$ which could only be removed upon request with the network administrators, causing some noise in the tail of the degree distribution.

The node-to-node distance distribution shown in Figure 6.4 demonstrates how the network adheres to the small-world property (see Section 6.2.3). This distribution was obtained by sampling $100,000$ node pairs $u, v \in V$, computing the value of distance $d(u, v)$, and then counting for each obtained distance value how frequently it was observed. This distance distribution was also used to the derive the average distance of 4.75 listed in Table 6.1. The radius and diameter of the graph were computed using the algorithms described in Chapter 2 and Chapter 4.

A set of nodes W of size $|W| = 4,867$ (0.06%) has been manually labeled by the network administrators as “prominent”. This subset consists of various Dutch politicians, artists, athletes and actors and will be considered as a ground truth for assessing the performance of different measures of prominence. We note that all prominent nodes are part of the giant component.

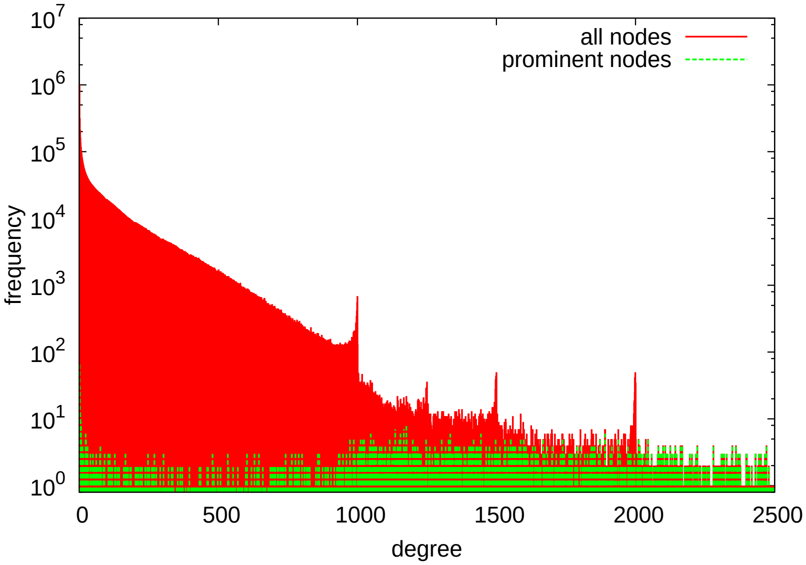

Figure 6.3: Degree distribution.

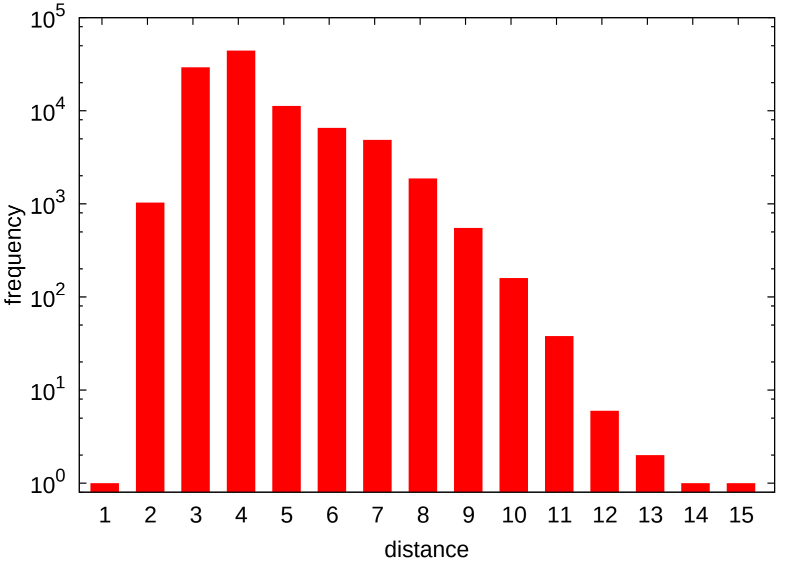

Figure 6.4: Distance distribution.

To the best of our knowledge, the only other study of a full snapshot of the full Hyves graph is provided in [35]. In this work, similar observations regarding the structural properties of the network are reported, and the distribution of various node attributes such as the age of the user are given.

6.6 Experiments

In this section we will compare the proposed algorithm to various existing approaches for determining node importance in networks. The algorithm as well as other discussed measures have been implemented in C++. Experiments were run on a 3.2GHZ machine with 10GB memory, allowing us to keep the large network dataset in memory. We start with a verification of the different node properties, after which we assess the performance of the BiasedRandomWalk algorithm.

6.6.1 Node properties

We have verified the two measures discussed in Section 6.4.1 on the discussed online social network dataset (see Section 6.5 for a description of the dataset). From the degree distribution shown in Figure 6.3, we can conclude that prominent users indeed have many more friends than regular users. A simple strategy for identifying prominent users would be to say that any user with more than for example $2,000$ friends is prominent. However, this would not only be incorrect because such a cut-off may be domain-specific and dependent on the type of social network, but it will also not help to identify the significant number of prominent users with anywhere between 0 and $2,000$ friends. For this degree range, there is also a (much larger) number of regular users with the same degree (notice the logarithmic vertical axis of Figure 6.3).

For the neighborhood density C_{nd}, we computed this value for $1,000$ randomly selected regular nodes and $1,000$ randomly selected prominent nodes, and found values of $1-0.131 = 0.869$ and $1-0.035 = 0.965$, respectively (the one minus notation is used to indicate the significant difference in the density summation of the measure of neighborhood density, see Section 6.4.1). This result is consistent with the intuition of regular users having a relatively more dense neighborhood than prominent users. Clearly, both of the properties that we discussed are related to the prominence of a user in the considered network.

6.6.2 BiasedRandomWalk

To verify the applicability of the proposed random walk algorithm, we first consider its convergence in terms of whether or not the set of identified prominent nodes becomes

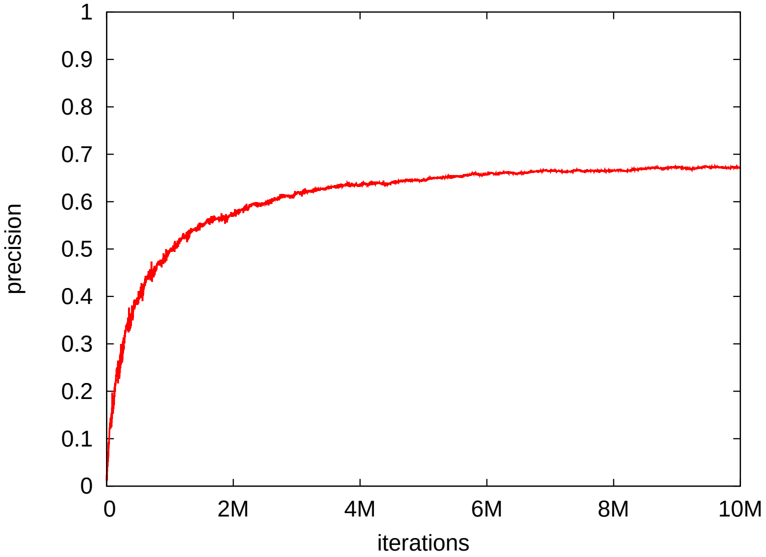

Figure 6.5: BIASEDRANDOMWALK convergence: number of iterations (horizontal axis) vs. precision (vertical axis) for a $n = 1$ million node sample of the original network.

consistent as the random walk algorithm runs. The result is displayed in Figure 6.5 for a 1 million node sample of the original graph. Clearly, the obtained value of the precision of the algorithm converges. We experimented with various sample sizes ($10,000$, $100,000$ and 1 million nodes) of the original 8 million node dataset, and consistently found that after $N = 10 \cdot n$ iterations, the precision did not show any significant improvement. Thus, for this network we conclude the parameter N can be set to $10 \cdot n$ to ensure a suitable result is obtained, which means that the running time would scale linearly with the number of nodes.

6.6.3 Results

The results of applying the various algorithms to the full friendship graph are outlined in Table 6.2. Here we compare the results of each of the methods based on $k = \ell$, meaning that we select exactly as many prominent people as there are in the dataset. Recall from Section 6.4 that we thus select the top $\ell = k$ nodes from the list of nodes sorted by their prominence function value. The BIASEDRANDOMWALK algorithm was executed with a budget of $N = 10 \cdot n = 81$ million iterations. Except for degree centrality which is fully deterministic, results are averaged over 10 runs in order to flatten the effect of outliers due to the inherent randomness of the approaches, resulting in standard deviations of less than 3% for each of the methods.

Measure	**Precision**	Time
Random	0.06%	0sec
HITS	51.4%	10min
PageRank	56.4%	10min
RandomWalk	63.9%	1min
NodeRanking	64.0%	1min
Degree Centrality	64.2%	0sec
BiasedRandomWalk	70.1%	13min

Table 6.2: Precision and indication of computation time of various importance measures with $k = \ell$.

As a baseline for comparison we could say that if we were to select ℓ random nodes from the complete set of nodes V to form the set W, we would on average find 0.06% of the prominent nodes in the network. Degree centrality, RandomWalk and the NodeRanking algorithm have roughly equal performance, and greatly improve upon this baseline by already identifying about 64.0% correctly. It turns out that HITS and PageRank performed significantly worse, which might be due to the fact that these three algorithms were at least initially designed for directed graphs. The proposed BiasedRandomWalk method improves another 6 percentage points upon degree centrality, the best performing existing method, demonstrating the advantage of looking at both the degree and the neighborhood density during the random walk.

As for the running time, obviously random selection and degree centrality require no additional computation time. PageRank and HITS both iterate over the set of edges 100 times to update the node values based on their neighboring nodes, each taking roughly 10 minutes in doing so. The random walk algorithms each run for $10 \cdot n$ steps, where in case of the plain random walk and the NodeRanking algorithm, no additional computation is done in each node. The BiasedRandomWalk method picks the neighbor with the highest C_{BRW} value which takes some computation time (but, assuming the graph is static, can be cached), running in little over 13 minutes in total.

One may argue that the number of prominent actors in a network is not always known in advance. Therefore we also did experiments in which we varied ℓ between $0.01 \cdot k$ and $2 \cdot k$ on the 1 million node sample of the full dataset, allowing the study of the precision, recall and F-measure curves. Assuming that we want to take a number of false positives for granted as is often permitted in practical applications, we may choose to focus solely on maximizing the recall value. Therefore, the recall value for each of the approaches as a function of the fraction of k is presented in Figure 6.6. In Figure 6.7 we furthermore present a comparison of the different F-values. At $0.75 \cdot k$, the F-value appears optimal for BiasedRandomWalk, and we would have a good

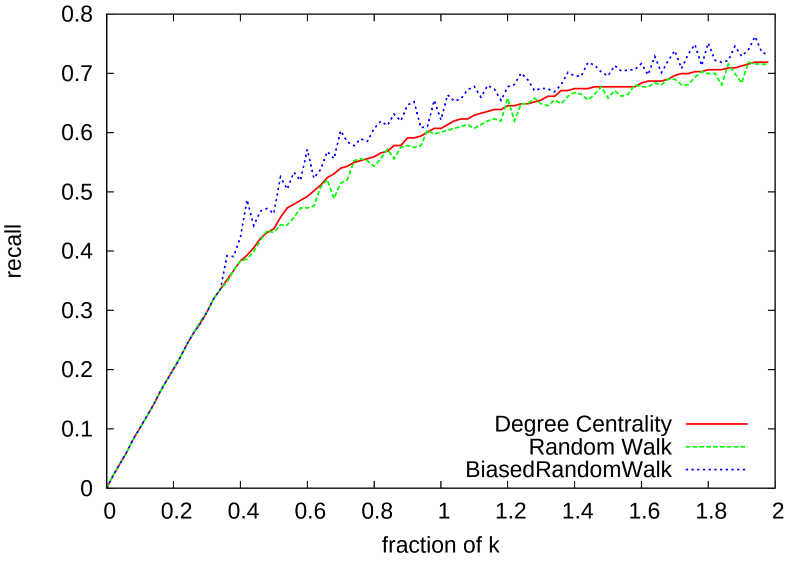

Figure 6.6: *Recall* for each of the methods.

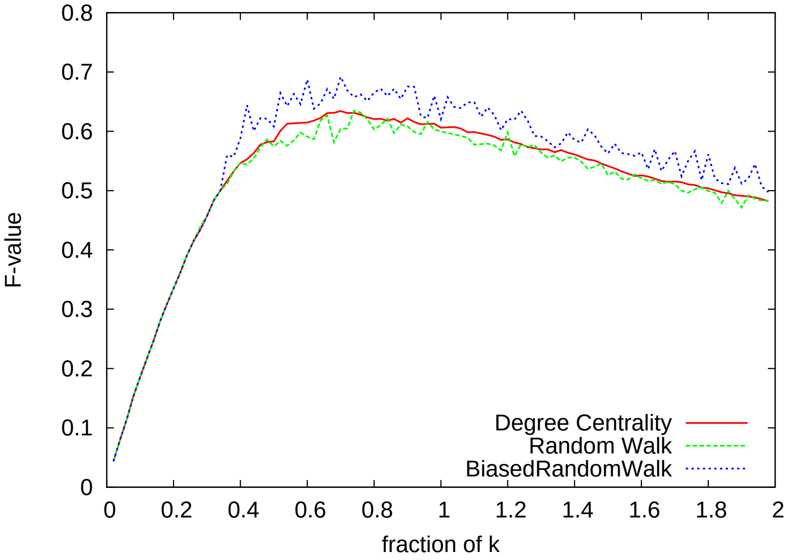

Figure 6.7: *F-value* for each of the methods.

balance between recall and precision. This optimum lies slightly lower around $0.70 \cdot k$ for the measure of degree centrality (we left out the other two identically performing measures because they both performed somewhat equal to degree centrality). Finally, note that for small values of ℓ (up to $0.3 \cdot k$), the obtained result is always perfect regardless of the method considered: apparently the top of the list is the same for each of the measures. It turns out these nodes simply had an enormously high degree (over $2,000$, see Figure 6.3), and were therefore selected by each of the methods. In general, we can conclude that the proposed BIASEDRANDOMWALK method works well, and is able to identify a significant portion of the prominent actors of the friendship graph of the considered online social network.

6.7 Conclusion

We have outlined various characteristic node properties of prominent actors in an online social network, and used these properties to create an algorithm for identifying prominent actors. Our algorithm, called BIASEDRANDOMWALK, combines the measures of degree centrality and neighborhood density in a random walk algorithm by having a bias towards nodes with high values for these two measures. Neighborhood density can be seen as a measure of the percentage of triadic closure, which is significantly lower for prominent actors as compared to regular nodes. On the other hand, the degree of prominent nodes is typically high. Experiments show that the proposed method works quite well, as standard centrality measures such as degree centrality, HITS and PageRank are outperformed in terms of precision, recall and F-measure.

In future work we would like to verify the extent to which the proposed random walk technique can be applied to other types of (social) networks, and how we can make the method parameter-free, or determine good parameter values based on properties of the network. We also want to consider the temporal aspect of importance, and study how properties of prominent nodes within a social network change over time.

Acknowledgment

We thank HYVES for making the structure of their (anonymized) friendship graph available.

Part II

Path Traversal Patterns

7

The Difficulty of Path Traversal in an Information Network

This chapter introduces a set of measures for determining the difficulty — for a human — of traversing paths in networks. The focus is on determining which node-based and path-based structural graph properties and measures say something about the difficulty of finding a certain path between two given nodes in a graph. Using a large corpus of over two million traversed paths on the online information network Wikipedia it is possible to demonstrate how the proposed techniques are able to accurately assess the human difficulty of finding a path between two given Wikipedia articles. The clickpaths analyzed in this chapter originate from the Wiki Game, an online game in which the main task is to connect two given random Wikipedia articles in as few clicks as possible. This chapter is based on:

- F. W. Takes and W. A. Kosters. The difficulty of path traversal in information networks. In *Proceedings of the International Conference on Knowledge Discovery and Information Retrieval (KDIR 2012)*, pages 138–144, 2012

7.1 Introduction

Searching and navigating through structured information such as Wikipedia, the web or a social network has become a common activity for many users. In this chapter we will analyze the way in which humans traverse structured data in search of a specific piece of information. The main goal of this study is to measure, understand and predict the difficulty of finding a path between two documents within a structured collection of information.

The motivation for this work comes from the idea that understanding the difficulty of path traversal may lead to a better understanding of human search behavior in general [62], which may in turn lead to improvements in the search strategy of an artificially intelligent search algorithm. Moreover, if we understand the aspects which complicate path traversal within structured data, then this information can possibly be used to improve the structure of the linked data itself [16].

Although search engines [40] can often help to find the content within a structured dataset that the user is looking for, sometimes search engine performance does not exactly meet the needs of the user [133]. This can happen for example because the user does not know the exact keyword that describes what he is looking for, because the search query was misinterpreted by the search engine, or because the required information is not indexed and is possibly located within the so-called Deep Web [58]. The Deep Web is the part of the internet which is not accessible to search engines, for example because the content resides within a database, because the pages are dynamic based on specific properties or settings of the user or because the content is only accessible from a limited range of machines.

When browsing for a piece of information within an information network, the user will have to reach the desired article by traversing the links that exist between the articles within the information network, forming a path towards the correct piece of information. We will study this type of path traversal by analyzing over two million paths traversed by (human) users of the well-known online encyclopedia *Wikipedia* (`http://www.wikipedia.org`). An advantage of studying paths on Wikipedia compared to for example clickstreams from the world wide web [11] is that Wikipedia contains much less “noise”, referring to to duplicate, false or untrusted information.

The Wikipedia paths analyzed in this chapter were gathered from the *Wiki Game* (`http://www.thewikigame.com`), a free online game in which the user is asked to connect two given random articles on Wikipedia. That is, starting from a certain source article, the main objective of the user is to reach the goal article by repeatedly following the clickable links within Wikipedia articles. This paper studies the difficulty of this particular task. If we are able to a priori determine the expected difficulty of a task, then this can be used to define multiple levels of difficulty for the Wiki Game.

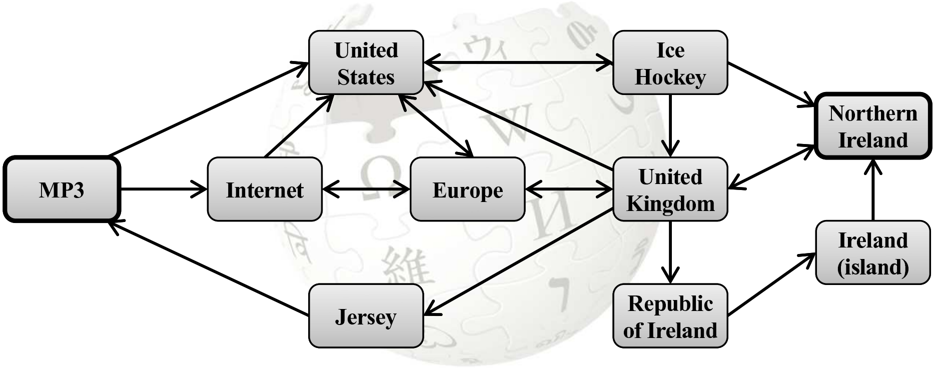

Figure 7.1: Subgraph of a (fictive) Wikipedia graph.

As an example of a path traversal task which is to be solved by a player of the Wiki Game, consider the path from the Wikipedia article on MP3 to the article on Northern Ireland. An actual (computed) shortest path of length 3 runs subsequently via the articles on the United States and Ice Hockey (see Figure 7.1). Human users attempting to find a path tend to know that Northern Ireland is somewhere in Europe, so from the article on MP3 they first find their way to an article related to Europe, for example via the page on the Internet which is a direct link from the article on MP3. Next, they will for example navigate to the article on the United Kingdom, from where they find the article on Northern Ireland. Some users take another detour on the way, for example via the page on the Republic of Ireland and the page on Ireland (island).

In turns out that humans, especially after some practice, are often able to link two given random articles on Wikipedia in less than ten clicks. This is actually a quite remarkable accomplishment, because even though a standard backtracking algorithm is certainly able to match or even beat humans in terms of path length, a human instead does not use millions of backtracking steps, but rather relies on background knowledge in terms of expected semantic relatedness [48] to find a path. Incorporating such extensive knowledge into an algorithm for classifying path difficulty, for example via ontologies [146], may in large information networks such as Wikipedia be very complex.

Throughout this chapter a range of node-based and path-based structural network properties and measures are proposed as indicators for the difficulty of connecting two articles. An advantage of considering structural features is that they may capture the direct relationship between the various concepts within the network, independent of which exact information network is studied. Also, structural properties are relatively easy to derive and compute, and do not require prior knowledge about the dataset. Moreover, while both the content as well as the linking structure of Wikipedia are subject to change, the classifiers that are proposed will only be affected by the second type of change, as article semantics are not considered. We will measure the quality of the proposed difficulty indicators by comparing their performance with the average human performance in terms of success or failure at completing a path.

The rest of this chapter is organized as follows. First, Section 7.2 discusses some notation, the various datasets used in this chapter and the main problem statement. We discuss related work in Section 7.3. Next we describe, analyze, test and compare the aspects which influence the difficulty of path traversal, at a node-based and a path-based scale, in Section 7.4 and 7.5, respectively. Finally, Section 7.6 concludes.

7.2 Preliminaries

In this section we discuss various concepts, definitions, notation and the considered datasets. Finally, we formulate the main problem statement and verification approach.

7.2.1 Concepts & definitions

The information network is represented by a directed graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ links. When we talk about a *path* between two nodes $u, v \in V$, we mean a sequence consisting of at least two nodes, starting at u and ending at v, where there is a link from each node to the next node in the sequence. A *shortest path* between two nodes $u, v \in V$ is a path of length $\ell \geq 1$ between u and v for which there is no other path from u to v of length smaller than ℓ. The length of such a shortest path, or in short the *distance*, is denoted by $d(u, v)$. Obviously, cycles may occur in paths, but not in shortest paths. Of course, it can happen that there are no (shortest) paths ($d(u, v) = \infty$) or that there are multiple (shortest) paths connecting two nodes. Because the graph is directed, it can happen that $d(u, v) \neq d(v, u)$. We define the *indegree* of a node $v \in V$ as the number of links pointing to node v, and similarly, the *outdegree* as the number of links pointing from node v to some other node.

7.2.2 Wikipedia

According to its own definition, "Wikipedia is a free, web-based, collaborative, multilingual encyclopedia project with over 3.9 million articles in English alone" (as observed in 2011). Considering solely the content of the articles and the links it contains, Wikipedia can be seen as a large directed graph, where each node represents an article, and each directed link a hyperlink within the source article pointing to the

Property	Value
Articles (n)	$3,464,902$
Directed links (m)	$82,019,786$
Largest WCC	99.9%
Average indegree	26
Average outdegree	22
Average distance ($\overline{d}$)	4.8
Effective diameter	7
Diameter	11

Table 7.1: Wikipedia dataset.

target article. In this study we will use the August 2011 English dataset of Wikipedia pagelinks from DBpedia version 3.7 (see [10] or `http://dbpedia.org`), from which we consider only the links to other Wikipedia articles, so we exclude links to external websites. Links to "special" articles such as articles describing a file, the category to which an articles belongs, or translations of the article, are also ignored. After some pruning and cleaning, the final Wikipedia graph that will be used for this study has statistics as presented in Table 7.1.

We note that the edge-to-node ratio, the diameter, defined as the length of a longest shortest path, the effective diameter (the 90-th percentile of the cumulative distribution of shortest path lengths), the average distance (between two nodes, sampled over $10,000$ node pairs) and the size of the largest weakly connected component (WCC) are consistent with that of other *small-world networks* [140]. The Wikipedia network furthermore has a power-law node degree distribution [149], suggesting that the Wikipedia graph indeed resembles other frequently studied real-world networks, such as the world wide web [12], internet topology networks [46] and social networks [76].

7.2.3 The Wiki Game

The Wiki Game is an online game launched in 2009, in which the user is assigned the task of connecting two given random articles on Wikipedia. Starting from a certain source article, the main objective is to reach the goal article by repeatedly clicking links on the page of the current article. While various types of games such as "Five clicks to Jesus", and "Six degrees of Wikipedia" exist, we will solely focus on "Speed Race" games, in which the task is to connect two given random Wikipedia articles in as few steps as possible, as quickly as possible, ultimately with a time limit of 120 seconds.

Property	**Value**
Tasks attempted	$407,268$
User-generated paths	$2,278,986$
Failed paths	72.0%
Successful paths	28.0%

Table 7.2: The Wiki Game dataset used in Chapter 7.

The Wiki Game dataset T consists of games (or *tasks*) and associated user-generated paths. A task $t \in T$ is essentially a (start, goal) pair (u, v) indicating between which two articles u and v (with $u, v \in V$) a path has to be formed. For each of these tasks we have a list of paths generated by the (fully anonymized) users that made an attempt at solving this task. These paths describe either a successful or a failed attempt at finding the goal article, and each have an associated path length. The data was filtered to exclude non-serious attempts (more than 40 clicks per task, or no clicks at all). This resulted in a dataset as presented in Table 7.2. Apparently, on average a task was performed by 5 to 6 users, and little less than one third of the total set of tasks presented to the users was successfully completed.

Figure 7.2 further clarifies the shortest path lengths of the tasks in the dataset, as well as the path length of the user-generated paths. We observe that even though shortest paths of length greater than 6 exist within Wikipedia, none of these tasks were included in the database of attempted tasks. Most tasks had a shortest path length somewhere between 2 and 4. Apparently, the average distance between the two pages composing a task is lower than the average distance in the entire Wikipedia dataset, indicating a small bias towards less obscure start and goal pages in the task database.

Figure 7.2 also shows how the distribution of the successful user-generated paths follows the same distribution as that of the shortest paths, but with an average path length that is roughly 2 times larger than the shortest path length (between 5 and 7), and a relatively fat tail. The distribution of the path length over all user-generated paths is clearly dominated by the failed paths, but as opposed to the successful paths, these distributions roughly follow a fat-tailed power law, indicating that when people “drop out” the path traversal process, they frequently do this early in the traversal process.

7.2.4 Problem definition

The main goal is to assess the difficulty of finding a path between two nodes in a directed graph:

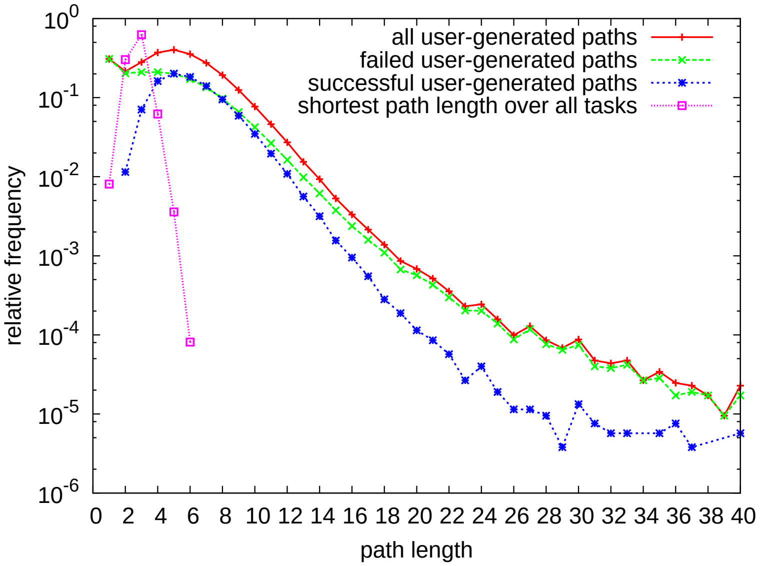

Figure 7.2: Relative frequency (vertical axis, logarithmic) of various path lengths (horizontal axis).

> Given a directed graph $G = (V, E)$ and nodes $u, v \in V$, can we assign a function value $f(u, v) \in [0; 1]$ indicating the difficulty of finding a path from u to v?

In this chapter we will consider various approaches (or *difficulty classifiers*) of assigning such a function value. We will evaluate the quality of an approach based on a comparison with the results obtained by the users on tasks from the Wiki Game. For each of the user-generated paths of a certain task $t \in T$ we know whether or not the path was successfully formed, allowing the definition of the average percentage of success $g(t) \in [0; 1]$ for task t. This information will serve as a ground truth for assessing the quality of the various difficulty classifiers.

Each difficulty classifier f can assign a function value $f(t)$ to all tasks $t \in T$, which allows us to create a partition $\{T_1, T_2, \ldots, T_q\}$ of the set of tasks T. The partitioning is done in such a way that the tasks within each T_i have the same function value (range), so that the (average) function value of the tasks in T_i is always greater than the average function value of the tasks in T_{i-1}, and where every T_i is maximal in size. The partitions can be used to define q different difficulty levels for the Wiki Game.

The overall quality of a classification measure will be determined by computing the *Pearson correlation coefficient* $c(f, g)$ of the classifier f and average percentage of success of the user-generated paths g, defined as:

$$c(f,g) = \frac{q\sum_i \overline{f(i)}\,\overline{g(i)} - \sum_i \overline{f(i)}\sum_i \overline{g(i)}}{\sqrt{q\sum_i \overline{f(i)}^2 - \left(\sum_i \overline{f(i)}\right)^2}\sqrt{q\sum_i \overline{g(i)}^2 - \left(\sum_i \overline{g(i)}\right)^2}}$$

Here, $\overline{f(i)}$ is equal to the average function value $f(t)$ of paths $t \in T_i$, and $\overline{g(i)}$ is the average percentage of success of the paths in T_i. As a second measure of comparison we will also consider the *Spearman rank correlation coefficient* $rc(f,g)$ of f and g, defined as:

$$rc(f,g) = \frac{\sum_i(\overline{f(i)} - \overline{f})(\overline{g(i)} - \overline{g})}{\sqrt{\sum_i \left(\overline{f(i)} - \overline{f}\right)^2 \sum_i \left(\overline{g(i)} - \overline{g}\right)^2}}$$

Here, $\overline{f}$ and $\overline{g}$ are equal to the average value over all i of $\overline{f(i)}$ and $\overline{g(i)}$, respectively. This coefficient measures the extent to which the relation between the classifier output and path difficulty can be described using a monotonic function. If we want a task at a certain difficulty level to always be harder than a task at the previous level, then we primarily aim for a high rank correlation coefficient.

In general, we will call a measure f correlated with path difficulty if it has a correlation greater than 0.8 (or smaller than -0.8) with the percentage of success g. For simplicity, we will denote the correlation coefficient and rank correlation coefficient by c and rc, respectively.

7.3 Related work

The structure behind Wikipedia has been analyzed in great detail, addressing tasks such as improving the linking structure [102] and automatic disambiguation of articles [63]. Furthermore, Wikipedia is frequently used as a knowledge base for external knowledge discovery tasks [138], and can serve as an excellent platform for computing (semantic) relatedness of concepts [48]. Patterns within clickpaths have also been analyzed extensively [24], and have proven useful for tasks such as page prediction [1, 119]. These patterns are often found within clickstreams from the web, where there is a great deal of “noise”, i.e., duplicate, false or untrusted information. On the web, information is frequently authored by one person or a very small group of people, whereas the number of participating users of Wikipedia is sufficiently large to counter spammers that spread for example false or biased information.

West and Leskovec [141] have compared human navigation in information networks such as Wikipedia with that of agents, using a dataset similar to the dataset studied in this chapter. They found that humans, when navigating within an information network, have expectations about what links should exist and base a high level reasoning plan upon this, and then use local information to navigate through the network. They furthermore mention that humans often miss “good” link opportunities on a page as their idea of semantic relatedness often overrules opportunistic clicking. In [142], the same authors show that progress in a goal-finding task is easiest far from and close to the target, with hubs being crucial in the beginning. To the best of our knowledge, the issue of path difficulty has so far not been addressed.

7.4 Node-based difficulty measures

In this section we consider *node-based* difficulty measures, by which we refer to properties that can be derived solely based on a node and its neighborhood (so, local information), in this case the Wikipedia article and its linked or linking articles. The advantage of such properties is that they are relatively easy to compute, and that they do not require knowledge about the entire dataset, which can be an advantage in extremely large datasets such as the world wide web or Wikipedia.

7.4.1 Degree measures

Having a large number of outgoing links for a certain node is likely to make it easier to directly reach a larger part of the graph from that particular node. Similarly, we expect that the number of incoming links of a node will probably make it relatively more easy to reach that node from any other node. We will verify the actual influence of these two measures of path difficulty by analyzing $q = 100$ ranges of the indegree of the goal article and the outdegree of the start article. The result is depicted in Figure 7.3, and a Bezier curve is drawn to better visualize the overall correlation. We observe no real significant correlation with the outdegree of the starting article ($c = 0.637$ and $rc = 0.789$). However, a strong correlation ($c = 0.850$ and $rc = 0.960$) is noticeable with respect to the indegree of the goal article and the actual percentage of success.

We can conclude from these results that the degree of the goal article is of significant influence to the difficulty of finding a certain path, whereas the degree of the starting node does not appear to play a notable role. An advantage of the node-based degree measure is that because the graph is stored as an adjacency list, the measure can be computed in $O(1)$.

7.4.2 Neighborhood measures

As the indegree is apparently a relevant indicator for the difficulty of finding a certain goal, it makes sense to refine this measure. Therefore we define the *h-neighborhood* $N_h(v)$ of a node $v \in V$ as the set of nodes with distance at most h from v, more specifically: $N_h(v) = \{w \in V \mid d(v,w) \leq h\}$. Similarly, we can define $N'_h(v) = \{u \in V \mid d(u,v) \leq h\}$, the *reverse neighborhood*, which is the set of all articles u with distance at most h to v. The *h-neighborhood size* is the number of nodes in the neighborhood of v, denoted by $|N_h(v)|$, and similarly we can define the *reversed h-neighborhood size* $|N'_h(v)|$. Obviously, 1-neighborhood size and reversed 1-neighborhood size are equal to the outdegree and indegree of a node plus 1 (the node itself), respectively.

We compared the neighborhood measures described above with path difficulty and found a significant correlation with the reversed 2-neighborhood size, which is essentially looking one step further than indegree. The functionality of this method can be explained by looking at the example graph in Figure 7.1. There, the article on Ice Hockey and the article on Ireland (island) both have an indegree of 1, while based on the degree of the neighbors, Ice Hockey seems much easier to reach than Ireland (island). This is nicely reflected by the reversed 2-neighborhood size, as $|N'_2(\text{Ireland (island)})| = 3$ and $|N'_2(\text{Ice Hockey})| = 6$, whereas the indegree would consider the

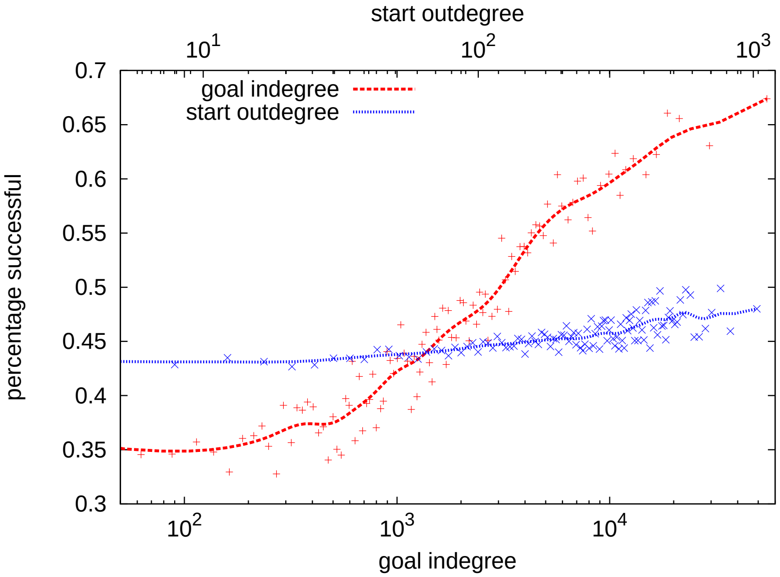

Figure 7.3: Start outdegree and goal indegree (horizontal axes, logarithmic) vs. percentage successful (vertical axis).

two nodes equally difficult to reach.

Figure 7.4 shows a plot of $q = 100$ intervals of the reversed 2-neighborhood of the goal article, again compared to the success percentage, and strong correlation coefficients ($c = 0.915$ and $rc = 0.978$) can be observed. Especially for the hardest tasks in the database ($g(t) < 0.35$), looking beyond the indegree helps to increase the amount of monotonicity.

In line with results from the previous section, the 2-neighborhood of the starting node did not appear to be correlated with the path difficulty ($c = 0.397$ and $rc = 0.492$). Furthermore we mention that, even though in some graphs it might make sense to look at (reverse) neighborhoods larger than $h = 2$, in the dense Wikipedia graph, considering more than the 2-neighborhood will quickly yield almost the entire graph, and indeed, correlation coefficients lower than 0.5 are observed when considering larger neighborhoods.

The neighborhood measures discussed in this subsection can be computed in $O((m/n)^{h-1})$ time per task. The average node indegree (or outdegree), (m/n), is between 20 and 30, still allowing for quick computation of the measure, especially in case of $h = 2$. So, the reversed 2-neighborhood size is a good indicator for path difficulty, whereas measures related to the degree of the starting article or neighborhood do not appear to be good at classifying path difficulty. This can be explained

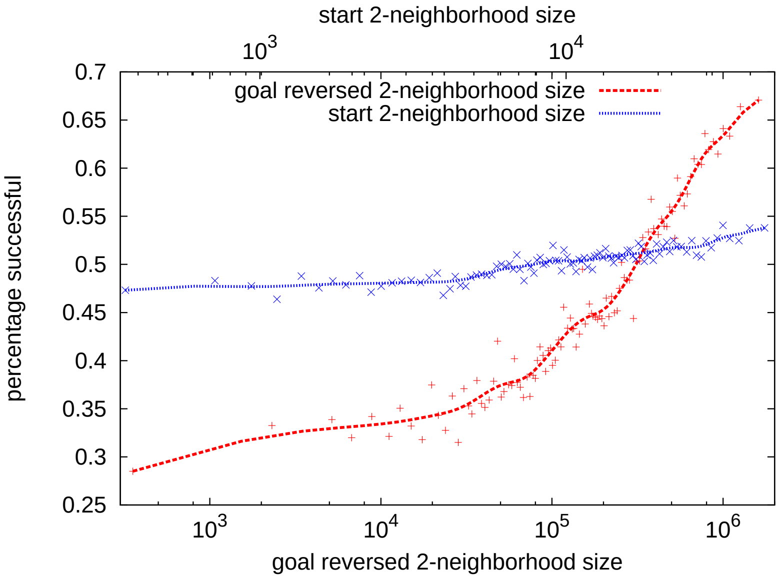

Figure 7.4: Start and goal 2-neighborhood measures (horizontal axes, logarithmic) vs. percentage successful (vertical axis).

by considering the small-world property of the Wikipedia: with relatively few steps it is possible to reach a large portion of the graph. It seems plausible that the starting node on its own is of little influence in general, because the user will often find his way to a hub-like node very quickly, from where the actual search for the goal node starts.

7.5 Path-based difficulty measures

In contrast with the previous section, we will now look at *path-based* properties, meaning that we look at actual paths between start and goal nodes in order to determine the difficulty of finding a path, possibly using global knowledge about the entire graph. Although the outcome in terms of difficulty prediction strength is expected to be higher, the computation time of path-based measures is longer: $O(m)$ per task.

7.5.1 Path length

When selecting two random articles on Wikipedia, due to the small-world property of the Wikipedia graph, the probability of selecting a pair of articles that is at a small distance of each other, is quite large. This is reflected in Figure 7.2, where the shortest path distribution of all played games is depicted, as well as the distribution of human formed path lengths of all successful paths. As mentioned in Section 7.2, the distribution of human path lengths does appear to have the same distribution shape as that of the actual shortest paths, suggesting a correlation between shortest path length and path difficulty.

Whereas we were able to aggregate the node-based measures from the previous section into $q = 100$ intervals, in case of path length we only have 6 different values. In Figure 7.6, the solid line shows for each actual *distance* (shortest path length) the percentage of successful human paths. This shows a strong correlation coefficient of $c = -0.957$ between the computed shortest path length and the percentage of successful paths, and an obvious rank correlation of $rc = -1.000$. However, a downside of considering distance as an indicator for difficulty is obviously the fact that it is only possible to define $q = 6$ different difficulty levels.

7.5.2 Number of shortest paths

We may also choose to look at the *number* of shortest paths $\sigma(u, v)$ between the start and goal article u and v. Intuitively, if there is only one shortest path from the start node to the end node, the task will be much harder compared to when there would have been thousands of shortest paths. Luckily, computing actual shortest path lengths is as easy as counting the number of shortest paths, as $\sigma(u,u) = 1$ and $\sigma(u,v) = \sum_{w \in B(u,v)} \sigma(u,w)$ with $B(u,v) = \{w \in N_1'(v) \mid d(u,v) = d(u,w) + 1\}$ [26]. The question is then how the number of shortest paths should be incorporated in a function value for assessing path difficulty. The number of shortest paths alone showed no significant correlation with path difficulty, which is understandable: a path of length 2 with 20 possible shortest paths is expected to be much easier to find than a path of length 4 with 20 shortest paths. So we propose to combine the distance with the number of shortest paths:

$$dsp(u,v) = d(u,v) + \alpha \left(1 - \frac{\log \sigma(u,v)}{\max_{w,z \in V}(\log \sigma(w,z))}\right)$$

The reason why we take the $\log$ of $\sigma(u,v)$ is motivated by Figure 7.5, where the plots of “shortest paths” indicate how the distribution of the number of shortest paths for each shortest path length decreases logarithmically. The parameter $\alpha \geq 0$ defines the amount of focus on the number of shortest paths as compared to the distance. If this parameter is set to 1, then a path of length 4 with only 1 possible shortest path is assumed to be easier to find than a path of length 5 with 2000 different shortest paths. Using linear parameter tuning with steps of 0.25, we obtained the best result for $\alpha = 1.5$, where we observe a strong correlation of $c = -0.895$ and $rc = -0.876$

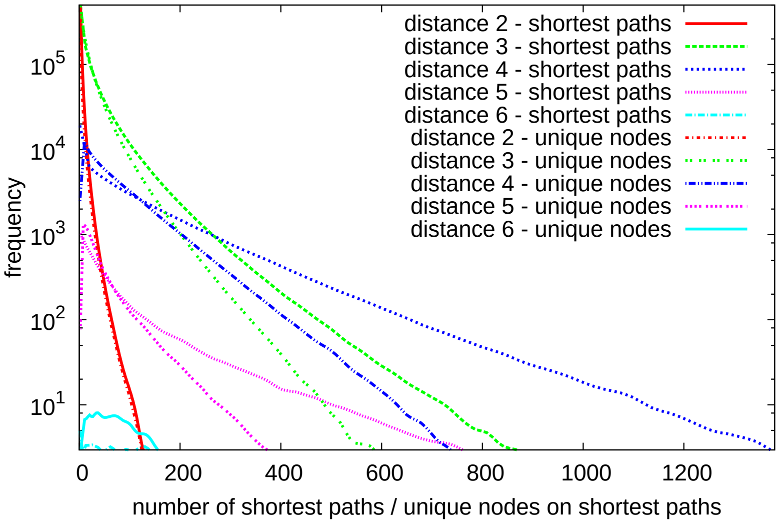

Figure 7.5: Frequency (vertical axis) of the number of shortest paths and number of unique nodes (horizontal axis) on these paths for each distance.

with path difficulty. The results are depicted in Figure 7.6.

7.5.3 Uniqueness of shortest paths

To further refine the measure from the previous section, we propose to look at the number of *distinct nodes* that occur within these shortest paths. This measure is based on the intuition that shortest paths quickly overlap, and that the extent to which paths overlap may influence the difficulty of a path finding task. For example, in Figure 7.1, the 3 shortest paths of length 3 from MP3 to United Kingdom run through a total of 4 different nodes: United States, Internet, Europe and Ice Hockey. The maximum number of unique nodes on 3 shortest paths of length 3 is 6 (3 times 2 unique intermediary nodes). Somewhat inspired by betweenness centrality, we propose to divide the number of nodes on the actual shortest paths by the maximum possible number of intermediary nodes, a measure which we will call *shortest paths uniqueness*. For the example, this results in a score of $\frac{4}{6} \approx 0.67$. We will incorporate this measure along with the distance in the difficulty classifier defined as:

$$dusp(u,v) = d(u,v) + \beta \left(1 - \frac{\log\left(\psi(u,v)\right)}{\log\left(d(u,v)\cdot\sigma(u,v)\right)}\right)$$

Here, $\psi(u,v)$ is a function that returns the number of distinct nodes on the shortest

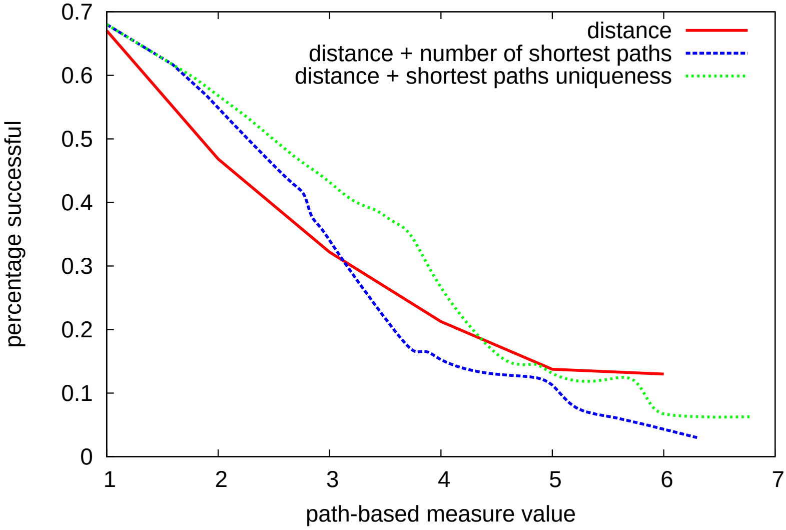

Figure 7.6: Various path-based measures (horizontal axis) vs. percentage successful (vertical axis).

Difficulty classifier	**Complexity**	q	c	rc
goal indegree	$O(1)$	100	0.850	0.960
start outdegree	$O(1)$	100	0.637	0.789
goal reversed 2-neighborhood size	$O(m/n)$	100	**0.915**	**0.978**
start 2-neighborhood size	$O(m/n)$	100	0.397	0.492
start-goal distance	$O(m)$	6	-0.957	-1.000
distance + number of shortest paths	$O(m)$	100	-0.895	-0.876
distance + shortest paths uniqueness	$O(m)$	100	$-$**0.924**	$-$**0.925**

Table 7.3: Summary of correlation coefficients (c), rank correlation coefficients (rc) and complexity (per task) of the proposed difficulty classifiers for q difficulty classes.

paths between u and v. The used values are again logarithmic as a result of the distribution of the number of unique nodes on the shortest paths, as depicted by the set of plots of "unique nodes" at various distances in Figure 7.5. The parameter $\beta \geq 0$ indicates the amount of focus on the number of distinct nodes over all shortest paths, and linear tuning of this parameter in steps of 0.25 showed that the best results were obtained for $\beta = 1.75$. The performance of the measure is displayed by the dotted line in Figure 7.6. We note that there are some "hickups" present in Figure 7.6, which might suggest that there is a better way of combining the two measures of distance and the uniqueness and number of shortest paths. The method nevertheless shows a correlation of $c = -0.924$ and $rc = -0.925$, demonstrating how shortest paths uniqueness does refine the path-based difficulty indicator from Section 7.5.1 based on the node-to-node distance.

7.6 Conclusion

Throughout this chapter we have proposed and analyzed the effectiveness of a range of techniques for classifying path traversal difficulty in information networks. The results are summarized in Table 7.3, in which the best-performing node-based and path-based measures are shown in bold.

With respect to the effectiveness of the various measures, we can generally say that node-based measures related to the goal article, such as the reversed neighborhood size, appear to be most effective, whereas node-based properties of the source article appear to be of little influence to path difficulty. In line with related work, we found that a user generally tends to quickly find his way to a hub node, from where the actual search process starts.

As for the path-based measures considered in this work, the distance between two articles is a good measure of difficulty, although as a result of the small-world property of Wikipedia, the range of different distances and thus the range of difficulty levels is very limited. Incorporating the number of shortest paths and the percentage of unique nodes over all shortest paths results in a path-based classifier with slightly better performance. However, a clear downside of the proposed path-based methods based on the number of shortest paths and their uniqueness, is that they require one parameter to be tuned. Furthermore, due to the higher complexity of path-based measures, one may favor the node-based classifiers in a practical application, such as in the Wiki Game. There, the difficulty classifiers outlined in this chapter could be used to improve the user experience by allowing users to select a desired difficulty level at which they want to play.

In future work we would like to improve our difficulty measures by including more article-specific information, such as the link density of the considered article. Furthermore, we want to analyze the frequent subpaths that exist both in the successful as well as in the failed paths created by humans. This information may help to obtain a better understanding of the search process of a certain user or group of similar users, possible allowing personalization of the difficulty indicators.

Acknowledgment

We thank Alex Clemesha, creator of the Wiki Game, for providing the data.

8 Mining User-Generated Path Traversal Patterns in an Information Network

This chapter studies patterns occurring in user-generated clickpaths within the online encyclopedia Wikipedia. The clickpath data originates from over seven million goal-oriented clicks gathered from the Wiki Game, an online game in which the goal is to find a path between two given random Wikipedia articles. First we propose to use node-based path traversal patterns to derive a new measure of node centrality, arguing that a node is central if it proves useful in navigating through the network. A comparison with centrality measures from literature is provided, showing that users generally "know" only a relatively small portion of the network, which they employ frequently in finding their goal, and that this set of nodes differs significantly from the set of central nodes according to various centrality measures. Next, we consider so-called frequent traversal graphs, i.e., graphs that arise from considering the nodes and edges of the top-k frequent path traversal patterns. We demonstrate how a small set of patterns is enough to obtain a subgraph with structural properties similar to that of the original graph, showing that users are able to identify a small yet efficient portion of the graph that is useful for successfully completing their navigation goals. This chapter is based on:

- F. W. Takes and W. A. Kosters. Mining user-generated path traversal patterns in an information network. In *Proceedings of the IEEE/ACM International Conference on Web Intelligence (WI 2013)*, pages 284–289, 2013

8.1 Introduction

A large part of the gigantic amount of information that is nowadays available is organized in some sort of *network* structure. Examples include the world wide web, an online social network or an information network such as Wikipedia. In these networks (or graphs), each node represents an entity or a piece of information, and each link represents a tie or relationship between two entities. An important task that human users perform on a daily basis, is *searching* for a piece of content within such a network. Although search engines can often assist the user in performing such a search task, *navigating* to the desired page by means of clicking the links between the nodes in the network is still a common activity, as sometimes search engine performance does not exactly meet the user's needs [133]. In such cases, the user will have to reach the correct page by traversing hyperlinks that exist between the pages in the network, forming a path towards the correct piece of information. Throughout this chapter we consider the task of mining *traversal patterns* that occur within these types of clickpaths. The obtained patterns are useful for understanding pathfinding strategies in networks, and may even be useful in getting a better understanding of human search behavior in general [62].

The data used in this chapter originates from the Wiki Game, an online game in which the main task is to link two given random pages on Wikipedia. Employing his perception of the structure of the network, a user has to find his way to the goal article by clicking the directed links that exist between the various articles in the Wikipedia graph, essentially generating a goal-oriented clickpath. We will consider a newer version of the Wiki Game dataset introduced in Chapter 7, containing more than one million clickpaths, comprising a total of seven million goal-oriented clicks on Wikipedia pages. It is important to note that these clicks are fundamentally different from simply counting the number of visits to a certain page, as these counts would for example also include visits that immediately reach the desired goal page, for example via a search engine. Instead, the clickpaths that we will study consist of Wikipedia pages and links between pages that were actually considered useful, by the user, in *traversing* the network.

We will use node-based traversal patterns to address a problem within the field of network analysis called *node centrality*, defined as the importance of a node within the network. So-called *centrality measures* are widely used to assess this issue of node centrality, and examples include PageRank [107] as well as centrality measures that originate from the field of social network analysis such as degree centrality, closeness centrality and betweenness centrality [26]. While the aforementioned centrality measures all employ the structure of the network to assess the importance of a node, none of them incorporates the human perception of the information incorporated in the network. As it is ultimately the user who is going to assess whether or not a page is actually relevant, one could say that it is not the structure of the network which should serve as the basis of the centrality measure, but it should instead be the user's perception of the network that is going to determine the importance of a node. It may very well be that certain structurally central nodes in the network are not considered important or useful by the user, and vice versa. Therefore we introduce a user-defined measure of centrality based on frequently traversed nodes, arguing that a page is important if it proves useful in navigating through the network. Especially in networks where the user perception of the data plays a central role, such as in the world wide web, or in an information network, we believe that a user-defined measure makes more sense than a conventional user-insensitive approach. Furthermore, we introduce the measure of subgraph centrality which determines the centrality of a group of connected nodes with respect to the rest of the network, allowing an experimental verification of the quality in terms of ease of navigation of the user-perceived central nodes.

The rest of this chapter is organized as follows. In Section 8.2 we discuss some definitions and introduce our dataset. After discussing related work in Section 8.3, we introduce node-based patterns and our user-defined measure of centrality in Section 8.4. In Section 8.5 we analyze different types of subgraphs derived from frequent traversal patterns, and we perform experiments demonstrating the successful use of these subgraphs by humans. Section 8.6 concludes the chapter and provides suggestions for future work.

8.2 Preliminaries

This section starts with some basic definitions regarding graphs and paths that will later on allow us to precisely define our path traversal patterns and various derived measures. We also describe the clickpath dataset to which we will later on apply our path traversal pattern mining techniques.

8.2.1 Wikipedia graph

We will model the information network Wikipedia as a directed graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ directed links between pairs of nodes. The indegree $indeg(v)$ of a node $v \in V$ is equal to the number of incoming links of v, and similarly $outdeg(v)$ denotes the number of outgoing links. We define a *path* as a vector p of visited nodes, where for each subsequent node pair $(v_i, v_{i+1}) \in p$ there exists a link $e = (v_i, v_{i+1}) \in E$ in the original graph G. The *path length* is then equal to the number of links that was traversed to get from the first to the last node in the path. We define the *distance* $d(u, v)$ as the length of the shortest path between nodes u and v, meaning the minimum number of links that has to be traversed to get from u to v. If there is no path between u and v, then $d(u, v) = \infty$. In such cases, the graph has multiple strongly connected components, meaning that some nodes are not reachable from every other node by considering the directed links between the nodes. This does not necessarily mean that there are multiple weakly connected components, which we define as maximal sets of nodes such that every node can reach every other node by means of traversing the links between the nodes, regardless of the direction of the link. For convenience in later definitions, we denote the number of shortest paths between two nodes by $\sigma(u, v)$, and the number of shortest paths from u to v that runs through node w as $\sigma_w(u, v)$.

In this research, we use a Wikipedia graph consisting of the pagelinks from the English version of DBpedia version 3.7 and 3.8 (see [10] or `http://dbpedia.org`), which were mined from the original Wikipedia datasets in 2011 and 2012. We mention that by only considering actual pagelinks and ignoring links to special pages or external websites, each page represents an actual piece of information within the information network. Although the used Wikipedia graph is a bit newer than the version presented in Table 7.1 in Chapter 7, some more pruning of “special” Wikipedia pages was done, resulting in a graph with $n = 3,416,126$ nodes and $m = 83,271,539$ directed links, and further statistics similar to what we presented in the previous chapter.

8.2.2 The Wiki Game dataset

The clickpath data used in this chapter is based on clicks made by users of the Wiki Game (`http://www.thewikigame.com`). In this game, users are assigned the task of connecting two given random articles on Wikipedia by traversing the links that exist between Wikipedia articles. For additional information and an example of this game, the reader is referred to Section 7.2.3 of Chapter 7. Compared to the previous chapter, we study a newer version of the Wiki Game dataset. Furthermore, the focus is on the actual completed clickpaths, and failed paths are ignored.

Property	**Value**
All user-generated paths	$3,219,641$
Clicks in all user-generated paths	$17,151,824$
Percentage successful	35.3%
Successful paths	$1,137,337$
Clicks in successful paths	$7,135,060$

Table 8.1: The Wiki Game dataset used in Chapter 8.

The dataset used in this chapter consists of clickpaths generated between 2009 and 2012, where one clickpath corresponds to a played game (or task), which is is essentially a (start, goal) pair between which a path has been formed. In this chapter, we will only consider paths with a length between 3 and 20, thus filtering out non-serious attempts. A total of $3,219,641$ paths was generated, consisting of $17,151,824$ clicks in total. Of these tasks, little over one third was successfully completed, which is the part of the dataset that we consider in this chapter. This results in a dataset of $1,137,337$ clickpaths consisting of a total of $7,135,060$ clicks. The statistics discussed above are summed up in Table 8.1. Figure 8.1 shows the relative frequency of the lengths of all user-generated paths, as well as that of the computed shortest path lengths of the tasks.

8.3 Related work

Path traversal patterns in a hyperlinked environment have been a popular subject of study since the introduction of the web [30]. A lot of work has been done on mining the top-k frequent traversal patterns [93], often by using algorithms from the field of frequent itemset mining [53]. With the enormous amount of web traffic taking place these days, studying path traversal patterns from a stream has also become a useful

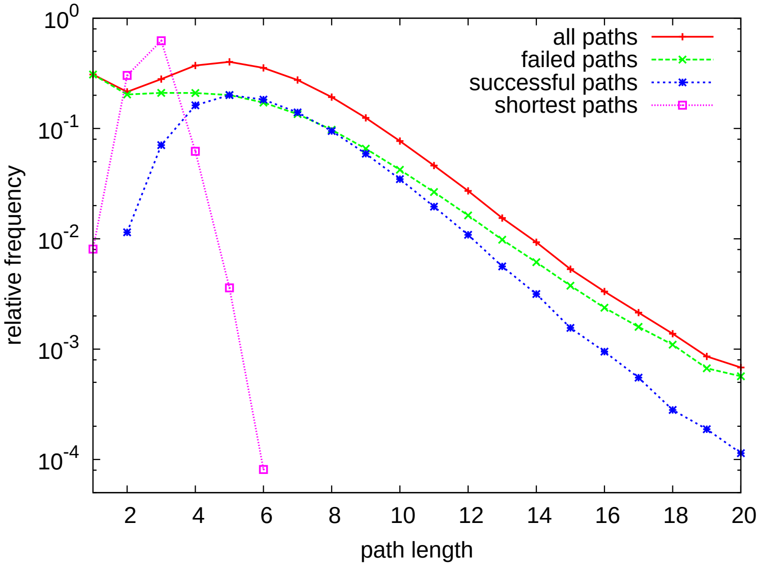

Figure 8.1: Relative frequency (vertical axis, logarithmic) of various path lengths (horizontal axis) of the filtered dataset.

task [92]. Most of the research in which clickpaths are analyzed within a confined environment considers weblogs from a particular website [1]. This chapter differs from such studies in a sense that all clicks in our dataset are goal-oriented and clicks are identifiable as one unique topic, namely the subject of the Wikipedia page. An overview of additional related work, for example on analysis of Wikipedia itself, can be found in Section 7.3.

In Chapter 7, we have investigated the difficulty of forming a path between two given random pages, showing that in Wiki Game, the indegree of the goal page as well as the reversed neighborhood, both local properties of the goal page, are good predictors of the difficulty of performing such a path traversal task. We have also demonstrated how the start page is of little influence as the user just navigates away from it quickly in search for a hub. Whereas the previous chapter only considered path traversal success or failure, in this chapter, we consider the patterns that arise from the actual clicks made by the users.

8.4 Path traversal patterns

In this section we will first introduce three types of path traversal patterns, after which we look in detail at node-based traversal patterns, and how these patterns can serve as a basis of a user-defined measure of centrality. We will compare this new measure with centrality measures from literature, that we briefly describe in Section 8.4.2.

8.4.1 Patterns

Given a dataset P consisting of a large number of clickpaths, we are interested in *patterns*, i.e., observable phenomena that occur more frequently than expected. For our clickpath dataset, it is possible to distinguish between the following frequencies in order to define our patterns:

- *Node traversal frequency*: the number of times a node v occurs in all paths p from P.

- *Edge traversal frequency*: the number of times an ordered pair of subsequent nodes (v_1, v_2) occurs in all paths p from P.

- *Subpath traversal frequency*: the number of times an ordered sequence of three or more subsequent nodes $(v_1, v_2, v_3, \ldots)$ appears in all paths p from P.

Obviously, relaxing the definition of subpath traversal frequency to length two or one, yields the definitions of respectively edge and node traversal frequency. Similar to the definitions often given in the area of frequent itemset mining [53], we call an observation *frequent* if it occurs more often than a certain threshold $\theta > 0$ amongst all paths, allowing the definition of our patterns: *frequent nodes*, *frequent edges* and *frequent subpaths*. For a given threshold θ, every node within a frequent edge and every edge within a frequent subpath, is also frequent. We define the set of top-k frequent patterns as the set of $k \geq 1$ patterns with the highest frequency, allowing us to again define derived sets called *top-k frequent nodes*, *top-k frequent edges* and *top-k frequent subpaths*. The most simple patterns based on frequent nodes are further discussed in this section, whereas graphs derived from more complex traversal patterns are considered in Section 8.5.

8.4.2 Centrality measures

Node centrality as the importance of a certain node in the graph. A centrality measure M returns the centrality $C_M(v)$ of a node $v \in V$. We consider the following (existing) centrality measures, somewhat ordered by their complexity in terms of computation time:

Indegree centrality

$$C_{indeg}(v) = \frac{indeg(v)}{n-1}$$

Closeness centrality

$$C_c(v) = \frac{1}{\frac{1}{n-1}\sum_{w \in V} d(v,w)}$$

PageRank

$$C_{PR}(v) = PR(v)$$

HITS

$$C_{HITS}(v) = a(v)$$

Betweenness centrality

$$C_b(u) = \sum_{\substack{v,w \in V \\ v \neq w, u \neq v, u \neq w}} \frac{\sigma_u(v,w)}{\sigma(v,w)}$$

A discussion and more elaborate definition of these measures is given in Section 5.4.1 and Section 6.3. Each of the centrality measures results in a number between 0 and 1, where a higher score indicates that the node is more central. For convenience, we normalize the centrality values such that the most central node has a centrality value of 1. Clearly, distance based measures do not perform well when there is more than one connected component. Therefore we will only consider the largest strongly connected component of the Wikipedia graph. We believe that we have covered the most common and applicable ones in this subsection, using similar arguments regarding the type of measures as presented in Section 5.4.1.

8.4.3 User-defined node centrality

Recall from Section 8.4.1 that considering the *top-k frequent nodes* means that if we sort the list of nodes by their node frequency value, we consider the k nodes with the highest frequency. For our clickpath dataset, this means that we are looking at the k nodes that were most frequently used to traverse the graph. This list is actually quite interesting, as it indicates which k nodes are considered important, by the user, in navigating through the graph. We use this data as a basis for our user-defined measure of centrality, proposing to count the number of clicks that an article v received (denoted by $clicks(v)$) and divide it by the total number of clicks made in order to obtain our user-defined measure of centrality:

User-defined centrality

$$C_{ud}(v) = \frac{clicks(v)}{\sum_{w \in V} clicks(w)}$$

To get an idea of the values returned by this function, Figure 8.2 shows the frequency of each node traversal count over all nodes in the graph. The distribution follows a clear power-law, meaning that many nodes are visited only a few times, and a few nodes are visited quite often. We are obviously interested in the tail of the distribution: the set of nodes that is visited very frequently.

8.4.4 Measure evaluation

Assessing the quality of a centrality measure is not a trivial task, and often comes down to simply comparing one centrality measure with another centrality measure. An alternative would be to have a subjective evaluation done by a human, and then determine the extent to which the ranking produced by the centrality measures resembles the user's perception of the importance of these nodes. An example of a more authoritative ground truth for centrality is provided in Chapter 6 in the context of online social networks, where celebrities have a special labeling created by the administrators of the network, reflecting the celebrity status of the real-world person behind the profile. However, often researchers rely on manual inspection of the top-k most central nodes [106], or simply compare their measure with other existing centrality measures [22]. If we are only interested in the relative ranking of entities in two top-k lists, measures such as Kendall's tau and Spearman's weighted footrule can be used [79].

In our experiments, we will use two different ways of comparing centrality measures, as suggested in [22] (though in a somewhat different setting). Often, a centrality measure is used to find the top-k most central nodes, and we mention that the evaluation techniques that we discuss here are designed such that thus only the top-k nodes are evaluated. A rather basic technique is to compare top-k nodes of two centrality measures and determine the percentage of nodes that overlap. For example, for $k = 1$, we simply verify whether the most central node is equal for both measures. We call this measure *top-k precision*, defined as follows:

$$\text{top-}k \text{ precision} = \frac{|A_k \cap B_k|}{k}$$

Here, $A_k, B_k \subseteq V$ represent the sets of top-k nodes returned by centrality measures A and B. Alternatively, when the actual centrality value of the top-k nodes is also of importance, we can look at the correlation between the centrality values in two lists of nodes. We call this measure *top-k correlation* and define it as the Pearson correlation coefficient between the centrality values of the two methods. Important to note here is that measure A is considered as the ground truth: we compare the centrality values of the top-k nodes of measure A with that of measure B.

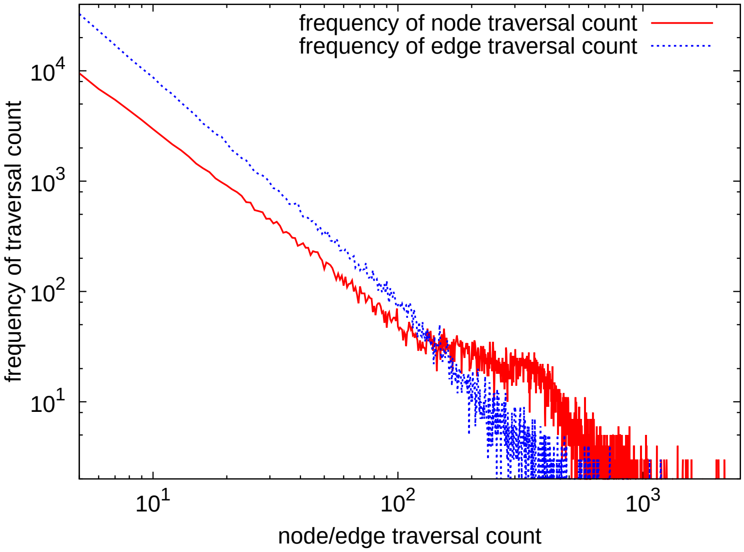

Figure 8.2: The frequency (vertical axis, logarithmic) of different node and edge traversal counts (horizontal axis, logarithmic).

8.4.5 Experiments

In this section we use the user-defined measure of node centrality introduced in Section 8.4.3 as a ground truth for comparing the centrality measures listed in Section 8.4.2. We compare the different measures up to $k = 250$, based on an evaluation using both top-k precision (see Figure 8.3 and Table 8.2) and top-k correlation (see Table 8.2).

We note that for small values of k, big deviations for the top-k precision measure can be observed, which is due to the fact that with a low value of k, one mismatch has a relatively high influence on the actual percentage. In our experiments we also found that it is important not to lose the directed aspect of the Wikipedia network, as otherwise overview pages containing listings of events or people will be ranked too high. This is also the reason why both outdegree centrality and the HITS algorithm using the hub score instead of the authority score did not produce meaningful results.

Looking at the performance of the different centrality measures in Table 8.2, we can generally conclude that PageRank gives not only the highest, but judging from Figure 8.3 also gives the most consistent results when top-k precision is considered. Indegree centrality is a good second choose if top-k correlation is important. We mention that for values greater than $k = 250$, a somewhat consistent precision is observed. Altogether, it appears that centrality measures are able to explain only roughly half

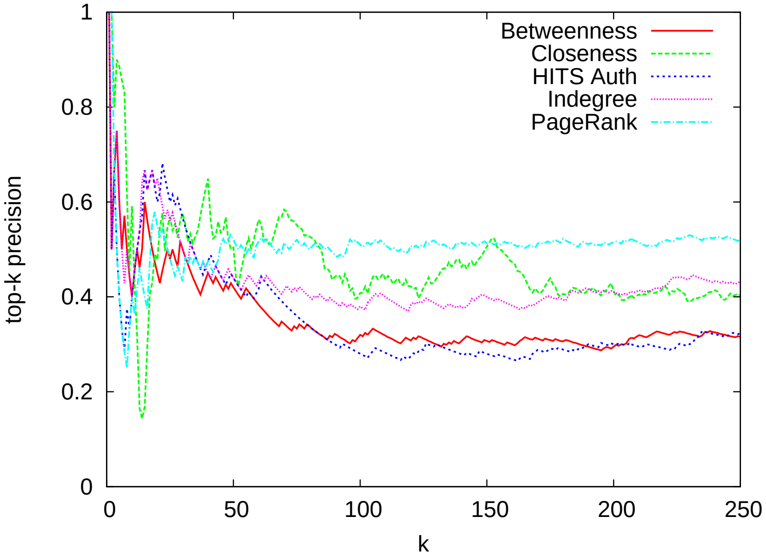

Figure 8.3: User-defined top-k precision (vertical axis) for different k (horizontal axis).

Measure	**Top-k precision**	**Top-k correlation**
User-defined	1.00	1.00
PageRank	0.51	0.76
Closeness	0.49	0.53
Indegree	0.37	0.83
Betweenness	0.32	0.71
HITS	0.28	0.62

Table 8.2: Comparison of centrality measures with user-defined centrality for $k = 100$.

of the nodes that are frequently used by humans to traverse the graph. This may lead us to believe that either humans are able to assess half of the central nodes in the graph, or that existing centrality measures are simply not able to produce the portion of nodes which is considered useful by the user. In the latter case, the only remaining question is then whether or not the set of nodes returned by the centrality measures is better or worse at ensuring that a large portion of the graph is easily reachable and thus useful for completing navigation goals. We will try to answer this question in the next section by looking at global properties of the path traversal patterns.

8.5 Global patterns

In this section we consider the global properties of the frequent patterns, creating subgraphs of the original network by considering the frequent node and edge traversal patterns.

8.5.1 Frequent traversal graphs

We define a *frequent traversal graph* as a graph consisting of frequent traversal patterns, distinguishing between two types of graphs:

- *Node-based frequent traversal graph*: the subgraph consisting of all nodes $v \in V$ and their connecting edges for which it holds that v is traversed more often than a certain threshold $\theta > 0$.

- *Edge-based frequent traversal graph*: the subgraph consisting of all links $(u, v) \in E$ and their node endpoints for which it holds that (u, v) is traversed more often than a certain threshold $\theta > 0$.

Analogously to the definitions given in Section 8.4.1, we can define top-k node-based and edge-based frequent traversal graphs, consisting of the top-k most frequently visited nodes and edges, respectively. Iterating over increasing values of k then yields node-based and edge-based evolving graphs.

Some properties of these two types of subgraphs are shown in Figure 8.4 and Figure 8.5 for respectively the frequent nodes and frequent edges of our Wikipedia clickpath dataset. We note that when considering frequent edges, the average distance between two nodes quickly (from roughly $k = 5,000$ onwards) resembles that of the original Wikipedia graph (4.55), and then remains surprisingly stable as k increases. The node-based frequent traversal graph does not resemble the distance distribution of the original graph, as this type of subgraph also contains many edges that were not actually traversed, but are simply present between the frequent nodes in the original graph, creating many more connections between the nodes than were actually traversed. Indeed, considering only the edge-based frequent patterns might make more sense, as the user apparently “knew” these exact links, and not just the nodes. The findings presented here may indicate that the user is able to select a representative portion of the edges (and by that a portion of the nodes). In the next section, an attempt is made to measure the effectiveness of this central portion of nodes.

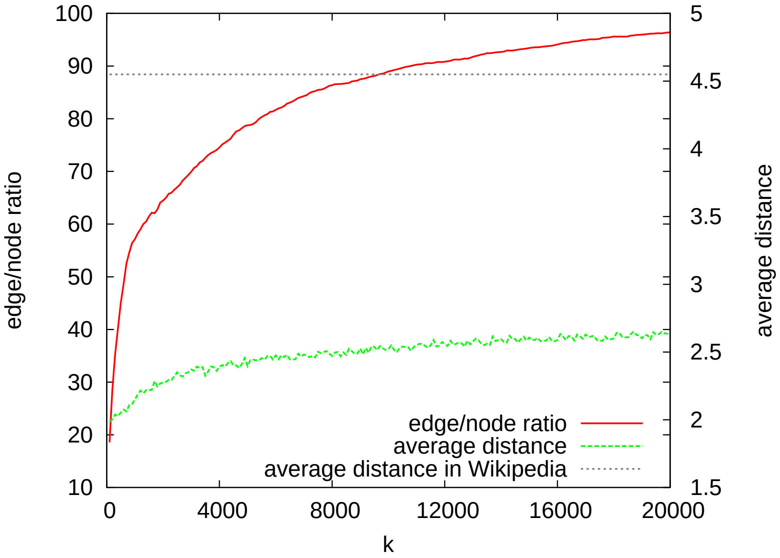

Figure 8.4: Values (vertical axes) of different properties of the node-based frequent traversal graph for different k (horizontal axis).

8.5.2 Subgraph centrality

The final question which we aim to answer in this chapter, is whether or not the frequent traversal graphs are actually better or worse than graphs derived from traditional centrality measures in terms of being able to quickly reach a large portion of the original graph, and thus ensuring ease of navigation. To do this, we introduce the measure of *subgraph centrality*, which we define as the centrality (according to some existing measure, in our case closeness centrality) of a *set* of nodes, namely the set of top-k nodes obtained through a centrality measure. To determine the centrality of this set of nodes, we merge the set of top-k frequent nodes into one node, realizing the equivalent of setting the weight of all edges between frequent nodes to zero.

In Figure 8.6 we show for increasing k the subgraph centrality values derived from the frequent nodes in the user-defined measure and the PageRank centrality measure. We have chosen to provide a comparison with PageRank and indegree because they performed best in terms of precision and correlation according to our experiments in Section 8.4.5. We observe how the subgraph centrality of the user-defined frequent traversal graph compares quite well to that of the PageRank subgraph, which indicates that the user is able to select a portion of nodes which in terms of reachability is equal to that of a centrality measure. For $k > 1,200$, the quality of the user-defined centrality is even higher than that of PageRank, suggesting that users are able to select a portion

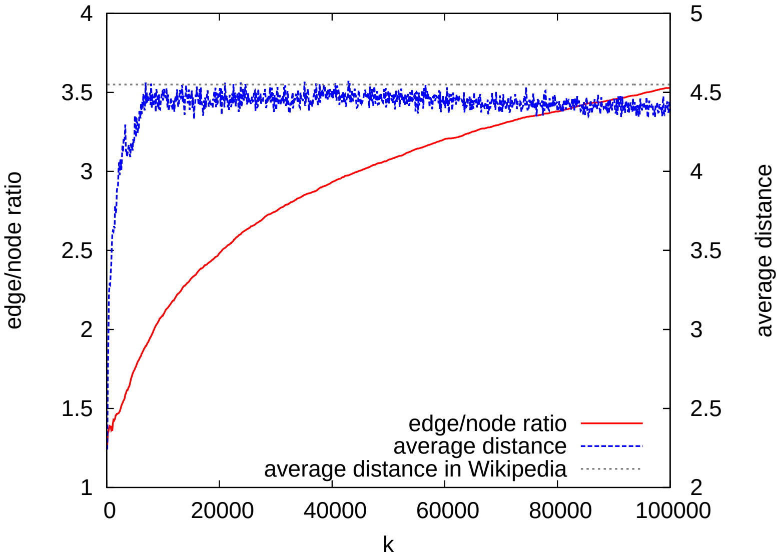

Figure 8.5: Values (vertical axes) of different properties of the edge-based frequent traversal graph for different k (horizontal axis).

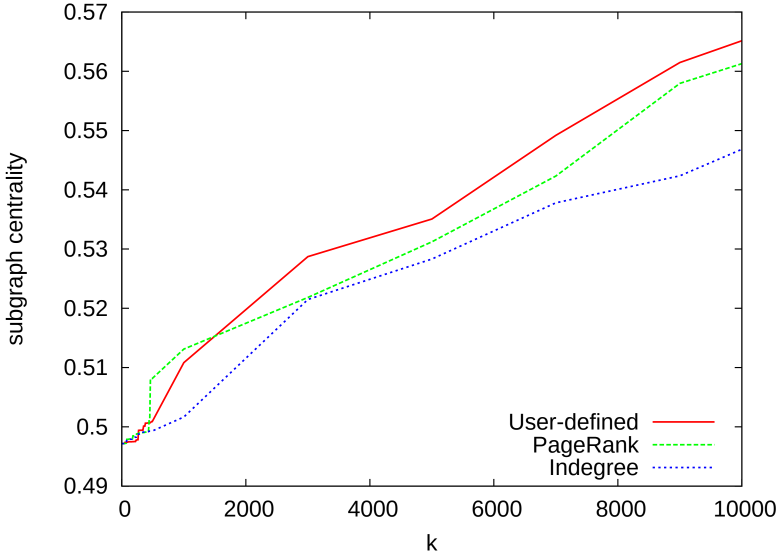

Figure 8.6: Comparison of subgraph centrality (vertical axis) of various centrality measures for different values of k (horizontal axis).

of the nodes of the graph which is better for realizing a low node-to-node distance than a traditional measure such as PageRank.

8.6 Conclusion

Throughout this chapter we have looked at mining path traversal patterns from the information network Wikipedia, aiming to understand and measure the quality in terms of navigation of user-generated traversal patterns. Using data gathered from over seven millions clicks made in the Wiki Game, we have derived a new measure of node centrality based on frequently traversed nodes.

It turns out that roughly half of the set of most frequently traversed nodes overlaps with the set of central nodes according to centrality measures such as PageRank. The additional nodes that are frequently visited by the users do appear to be useful, which we have demonstrated by using frequent traversal graphs and the notion of subgraph centrality. The subgraphs that can be derived from the frequently traversed nodes appear to be more central than the set of nodes derived from an existing centrality measure. This shows how users are apparently able to select an efficient portion of the graph that is useful in traversing the graph, specifically realizing a short distance to all other nodes in the graph. Although we have shown that the user is able to select an efficient subset of the graph for completing navigation goals, it remains an open question exactly *how* the user selected this subset. Clearly, a subset derived using a centrality measure or a random subset performs similar or worse, so from an artificial intelligence point of view, the performance of the user is quite remarkable.

In future work, we want to see if we can extend the study of traversal patterns to more complex patterns based on frequent edges, possibly to study edge centrality [100], or based on frequent (interleaved) subpaths. Last but not least, the topics and techniques discussed in this chapter can possibly be extended to other types of graphs such as social networks, in which frequently traversed nodes and edges may indicate important actors and ties in the network.

Acknowledgment

We thank Alex Clemesha, creator of the Wiki Game, for providing the data.

Bibliography

[1] R. Agarwal, K. Veer Arya, and S. Shekhar. An architectural framework for web information retrieval based on user's navigational pattern. In *Proceedings of the International Conference on Industrial and Information Systems (ICIIS)*, pages 195–200, 2010.

[2] Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *Proceedings of the 16th International World Wide Web Conference (WWW 2007)*, pages 835–844, 2007.

[3] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.

[4] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD 2013)*, pages 349–360, 2013.

[5] T. Akiba, C. Sommer, and K. Kawarabayashi. Shortest-path queries for complex networks: Exploiting low tree-width outside the core. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT 2012)*, pages 144–155, 2012.

[6] R. Albert. Scale-free networks in cell biology. *Journal of Cell Science*, 118:4947–4957, 2005.

[7] R. Albert and A. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47, 2002.

[8] R. Albert, H. Jeong, and A. Barabási. Internet: Diameter of the world-wide web. *Nature*, 401(6749):130–131, 1999.

[9] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997.

[10] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A nucleus for a web of open data. In *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. 2007.

[11] A. Banerjee and J. Ghosh. Clickstream clustering using weighted longest common subsequences. In *Proceedings of the SIAM Workshop on Web Mining*, pages 33–40, 2001.

[12] A. Barabási, R. Albert, and H. Jeong. Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A: Statistical Mechanics and its Applications*, 281(1):69–77, 2000.

[13] A. Barabási, H. Jeong, Z. Néda, E. Ravasz, A. Schubert, and T. Vicsek. Evolution of the social network of scientific collaborations. *Physica A: Statistical Mechanics and its Applications*, 311(3-4):590–614, 2002.

[14] V. Batagelj and A. Mrvar. Pajek datasets. Accessed February 28, 2014. `http://vlado.fmf.uni-lj.si/pub/networks/data`.

[15] K. Batool and M. A. Niazi. Towards a methodology for validation of centrality measures in complex networks. *PloS ONE*, 9(4), article e90283, 2014.

[16] C. Bizer, T. Heath, and T. Berners-Lee. Linked data: The story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[17] K. Boitmanis, K. Freivalds, P. Ledi, and R. Opmanis. Fast and simple approximation of the diameter and radius of a graph. In *Experimental Algorithms*, volume 4007 of *Lecture Notes in Computer Science*, pages 98–108. 2006.

[18] P. Boldi, M. Rosa, and S. Vigna. HyperANF: Approximating the neighbourhood function of very large graphs on a budget. In *Proceedings of the 20th International Conference on World Wide Web (WWW 2011)*, pages 625–634, 2011.

[19] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *Proceedings of the 13th International World Wide Web Conference (WWW 2004)*, pages 595–601, 2004.

[20] K. D. Bollacker, S. Lawrence, and C. L. Giles. Citeseer: An autonomous web agent for automatic retrieval and identification of interesting publications. In *Proceedings of the 2nd International Conference on Autonomous Agents*, pages 116–123, 1998.

[21] M. Borassi, P. Crescenzi, M. Habib, W. A. Kosters, A. Marino, and F. W. Takes. On the solvability of the six degrees of Kevin Bacon game — A faster graph diameter and radius computation method. In *Proceedings of the 7th International Conference on Fun with Algorithms (FUN 2014)*, volume 8496 of *Lecture Notes in Computer Science*, pages 52–63. 2014.

[22] S. P. Borgatti, K. M. Carley, and D. Krackhardt. On the robustness of centrality measures under conditions of imperfect data. *Social Networks*, 28(2):124–136, 2006.

[23] S. P. Borgatti and M. G. Everett. A graph-theoretic perspective on centrality. *Social Networks*, 28(4):466–484, 2006.

[24] J. Borges and M. Levene. Data mining of user navigation patterns. In *Web Usage Analysis and User Profiling*, volume 1836 of *Lecture Notes in Computer Science*, pages 92–112. 2000.

[25] D. Boyd and N. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1):210–230, 2007.

[26] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[27] U. Brandes and C. Pich. Centrality estimation in large networks. *International Journal of Bifurcation and Chaos*, 17(7):2303–2318, 2007.

[28] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer networks*, 33(1):309–320, 2000.

[29] F. Buckley and F. Harary. *Distance in Graphs*. Addison-Wesley, 1990.

[30] M.-S. Chen, J. S. Park, and P. S. Yu. Efficient data mining for path traversal patterns. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):209–221, 1998.

[31] P. Chen, H. Xie, S. Maslov, and S. Redner. Finding scientific gems with Google's PageRank algorithm. *Journal of Informetrics*, 1(1):8–15, 2007.

[32] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: User movement in location-based social networks. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (KDD 2011)*, pages 1082–1090, 2011.

[33] D. J. Cook and L. B. Holder. *Mining Graph Data*. John Wiley & Sons, 2006.

[34] D. G. Corneil, F. F. Dragan, and E. Köhler. On the power of BFS to determine a graph's diameter. *Networks*, 42(4):209–222, 2003.

[35] R. Corten. Composition and structure of a large online social network in the Netherlands. *PloS ONE*, 7(4), article e34760, 2012.

[36] P. Crescenzi, R. Grossi, M. Habib, L. Lanzi, and A. Marino. On computing the diameter of real-world undirected graphs. *Theoretical Computer Science*, 514:84–95, 2013.

[37] P. Crescenzi, R. Grossi, C. Imbrenda, L. Lanzi, and A. Marino. Finding the diameter in real-world graphs. In *Proceedings of the 18th Annual European Symposium on Algorithms (ESA 2010)*, pages 302–313, 2010.

[38] P. Crescenzi, R. Grossi, L. Lanzi, and A. Marino. A comparison of three algorithms for approximating the distance distribution in real-world graphs. In *Theory and Practice of Algorithms in (Computer) Systems*, volume 6595 of *Lecture Notes in Computer Science*, pages 92–103. 2011.

[39] P. Crescenzi, R. Grossi, L. Lanzi, and A. Marino. On computing the diameter of real-world directed (weighted) graphs. In *Experimental Algorithms*, volume 7276 of *Lecture Notes in Computer Science*, pages 99–110. 2012.

[40] W. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley, 2010.

[41] G. H. Dal, W. A. Kosters, and F. W. Takes. Fast diameter computation of large sparse graphs using GPUs. In *Proceedings of the 22nd IEEE International Conference on Parallel, Distributed and Network-based Processing (PDP 2014)*, pages 632–639, 2014.

[42] P. Desikan and J. Srivastava. Mining temporally evolving graphs. In *Proceedings of the 6th WEBKDD Workshop in conjunction with the 10th ACM International Conference on Knowledge Discovery and Data Mining (KDD 2004)*, 10 pages, 2004.

[43] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[44] D. Dor, S. Halperin, and U. Zwick. All pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

[45] D. Eppstein and J. Wang. Fast approximation of centrality. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA 2011)*, pages 228–229, 2001.

[46] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *ACM SIGCOMM Computer Communication Review*, 29:251–262, 1999.

[47] L. Fu and J. Deng. Graph calculus: Scalable shortest path analytics for large social graphs through core net. In *Proceedings of the IEEE/ACM International Conference on Web Intelligence (WI 2013)*, pages 417–424, 2013.

[48] E. Gabrilovich and S. Markovitch. Computing semantic relatedness using Wikipedia-based explicit semantic analysis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, volume 7, pages 1606–1611, 2007.

[49] L. Getoor and C. P. Diehl. Link mining: A survey. *ACM SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

[50] J. Golbeck and J. Hendler. Accuracy of metrics for inferring trust and reputation in semantic web-based social networks. In *Lecture Notes in Computer Science*, volume 3257, pages 116–131, 2004.

[51] A. V. Goldberg, H. Kaplan, and R. F. Werneck. Reach for A*: Efficient point-to-point shortest path algorithms. In *Proceedings of the SIAM Workshop on Algorithm Engineering and Experiments (ALENEX 2006)*, pages 129–143, 2006.

[52] V. Gómez, A. Kaltenbrunner, and V. López. Statistical analysis of the social network and discussion threads in Slashdot. In *Proceedings of the 17th ACM International Conference on World Wide Web (WWW 2008)*, pages 645–654, 2008.

[53] G. Grahne and J. Zhu. Fast algorithms for frequent itemset mining using FP-trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(10):1347–1362, 2005.

[54] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large graphs. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM 2010)*, pages 499–508, 2010.

[55] A. Gubichev and T. Neumann. Path query processing on very large RDF graphs. In *Proceedings of the 14th International Workshop on the Web and Databases (WebDB 2011)*, 6 pages, 2011.

[56] P. Hage and F. Harary. Eccentricity and centrality in networks. *Social Networks*, 17:57–63, 1995.

[57] P. Harish and P. Narayanan. Accelerating large graph algorithms on the GPU using CUDA. In *High Performance Computing (HiPC 2007)*, volume 4873 of *Lecture Notes in Computer Science*, pages 197–208. 2007.

[58] B. He, M. Patel, Z. Zhang, and K. Chang. Accessing the deep web. *Communications of the ACM*, 50(5):94–101, 2007.

[59] E. M. Heemskerk, F. Daolio, and M. Tomassini. The community structure of the European network of interlocking directorates 2005–2010. *PLoS ONE*, 8(7), article e68581, 2013.

[60] P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004.

[61] O. Hinz, B. Skiera, C. Barrot, and J. U. Becker. Seeding strategies for viral marketing: an empirical comparison. *Journal of Marketing*, 75(6):55–71, 2011.

[62] I. Hsieh-Yee. Research on web search behavior. *Library & Information Science Research*, 23(2):167–185, 2001.

[63] J. Hu, G. Wang, F. Lochovsky, J. Sun, and Z. Chen. Understanding user's query intent with Wikipedia. In *Proceedings of the 18th International Conference on World Wide Web (WWW 2009)*, pages 471–480, 2009.

[64] H. Jeong, S. P. Mason, A. Barabási, and Z. N. Oltvai. Lethality and centrality in protein networks. *Nature*, 411(6833):41–42, 2001.

[65] R. Jin, N. Ruan, Y. Xiang, and V. Lee. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD 2012)*, pages 445–456, 2012.

[66] S. Jin and A. Bestavros. Small-world characteristics of internet topologies and implications on multicast scaling. *Computer Networks*, 50(5):648–666, 2006.

[67] B. H. Junker and F. Schreiber. *Analysis of Biological Networks*. John Wiley & Sons, 2008.

[68] U. Kang, C. E. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec. Hadi: Mining radii of large graphs. *ACM Transactions on Knowledge Discovery from Data*, 5(2):8, 2011.

[69] A. M. Kentsch, W. A. Kosters, P. van der Putten, and F. W. Takes. Exploratory recommendations using Wikipedia's linking structure. In *Proceedings of 20th Belgium Netherlands Conference on Machine Learning (Benelearn 2011)*, pages 61–68, 2011.

[70] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[71] J. Kleinberg. The small-world phenomenon: An algorithm perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC 2000)*, pages 163–170, 2000.

[72] J. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and embedding using small sets of beacons. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS 2004)*, pages 444–453, 2004.

[73] B. Klimt and Y. Yang. The Enron corpus: A new dataset for email classification research. In *Machine Learning: ECML 2004*, volume 3201 of *Lecture Notes in Computer Science*, pages 217–226. 2004.

[74] KONECT. Koblenz network collection. Accessed February 28, 2014. `http://konect.uni-koblenz.de/networks/`.

[75] G. Kossinets and D. Watts. Empirical analysis of an evolving social network. *Science*, 311(5757):88–90, 2006.

[76] R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (KDD 2006)*, 2006.

[77] P. D. Kusuma, D. Radosavljevik, F. W. Takes, and P. van der Putten. Combining customer attribute and social network mining for prepaid mobile churn prediction. In *Proceedings of 22th Belgian Netherlands Conference on Machine Learning (Benelearn 2013)*, pages 50–58, 2013.

[78] C. Lampe, N. Ellison, and C. Steinfield. A Face(book) in the crowd: Social searching vs. social browsing. In *Proceedings of the 20th Anniversary Conference on Computer Supported Cooperative Work*, pages 167–170, 2006.

[79] A. Langville and C. Meyer. *Who's #1? The Science of Rating and Ranking*. Princeton University Press, 2012.

[80] A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2006.

[81] G. Laporte. Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416, 2009.

[82] LASAGNE. Laboratory of algorithms, models and analysis of graphs and networks. Accessed February 28, 2014. `http://piluc.dsi.unifi.it/lasagne`.

[83] J. Leskovec, L. Adamic, and B. Huberman. The dynamics of viral marketing. *ACM Transactions on the Web*, 1(1):5, 2007.

[84] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (KDD 2006)*, pages 631–636, 2006.

[85] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In *Proceedings of the 19th International Conference on World Wide Web (WWW 2010)*, pages 641–650, 2010.

[86] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the 11th ACM International Conference on Knowledge Discovery in Data Mining (KDD 2005)*, pages 177–187, 2005.

[87] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data*, 1(1), article 2, 2007.

[88] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[89] J. Leskovec, K. J. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th International World Wide Web Conference (WWW 2010)*, pages 631–640, 2010.

[90] L. Lesniak. Eccentric sequences in graphs. *Periodica Mathematica Hungarica*, 6:287–293, 1975.

[91] A. Levitin. *Introduction to the Design and Analysis of Algorithms*. Addison-Wesley, 3rd edition, 2011.

[92] H.-F. Li, S.-Y. Lee, and M.-K. Shan. On mining webclick streams for path traversal patterns. In *Proceedings of the 13th ACM International World Wide Web Conference (WWW 2004)*, pages 404–405, 2004.

[93] H.-F. Li, S.-Y. Lee, and M.-K. Shan. DSM-TKP: Mining top-k path traversal patterns over web click-streams. In *Proceedings of the IEEE/ACM International Conference on Web Intelligence (WI 2005)*, pages 326–329, 2005.

[94] M. Luiten, W. A. Kosters, and F. W. Takes. Topical influence on Twitter: A feature construction approach. In *Proceedings of 24th Benelux Conference on Artificial Intelligence (BNAIC 2012)*, pages 139–146, 2012.

[95] C. Magnien, M. Latapy, and M. Habib. Fast computation of empirically tight bounds for the diameter of massive graphs. *ACM Journal of Experimental Algorithmics*, 13, article 10, 2009.

[96] D. Magoni and J. Pansiot. Analysis of the autonomous system network topology. *ACM SIGCOMM Computer Communication Review*, 31(3):26–37, 2001.

[97] D. Magoni and J. Pansiot. Internet topology modeler based on map sampling. In *Proceedings of the 7th International Symposium on Computers and Communications (ISCC 2002)*, pages 1021–1027, 2002.

[98] D. Magoni and J.-J. Pansiot. Analysis and comparison of internet topology generators. In *NETWORKING 2002: Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications*, volume 2345 of *Lecture Notes in Computer Science*, pages 364–375. 2002.

[99] A. Marino. *Algorithms for Biological Graphs: Analysis and Enumeration*. PhD thesis, Università degli Studi di Firenze, 2012.

[100] P. D. Meo, E. Ferrara, G. Fiumara, and A. Ricciardello. A novel measure of edge centrality in social networks. *Knowledge-Based Systems*, 30:136–150, 2012.

[101] D. Merrill, M. Garland, and A. Grimshaw. Scalable GPU graph traversal. *ACM SIGPLAN Notices*, 47(8):117–128, 2012.

[102] D. Milne and I. H. Witten. Learning to link with Wikipedia. In *Proceedings of the 17th ACM International Conference on Information and Knowledge Management (CIKM 2008)*, pages 509–518, 2008.

[103] A. Mislove, M. Marcon, K. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM International Conference on Internet Measurement*, pages 29–42, 2007.

[104] M. E. Newman. Clustering and preferential attachment in growing networks. *Physical Review E*, 64(2):025102, 2001.

[105] T. O'Guinn, C. Allen, and R. Semenik. *Advertising and Integrated Brand Promotion*. Cengage Learning, 2011.

[106] T. Opsahl, F. Agneessens, and J. Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks*, 32(3):245–251, 2010.

[107] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. *Technical Report, Stanford Digital Library Technologies Project*, 1998.

[108] G. Palla, I. J. Farkas, P. Pollner, I. Derenyi, and T. Vicsek. Directed network modules. *New Journal of Physics*, 9(6):186, 2007.

[109] C. Palmer, P. Gibbons, and C. Faloutsos. ANF: A fast and scalable tool for data mining in massive graphs. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (KDD 2002)*, pages 81–90, 2002.

[110] M. Parnas and D. Ron. Testing the diameter of graphs. *Random Structures & Algorithms*, 20(2):165–183, 2002.

[111] G. Pavlopoulos, M. Secrier, C. Moschopoulos, T. Soldatos, S. Kossida, J. Aerts, R. Schneider, and P. Bagos. Using graph theory to analyze biological networks. *BioData Mining*, 4, 10 pages, 2011.

[112] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. In *Proceedings of the 18th ACM International Conference on Information and Knowledge Management (CIKM 2009)*, pages 867–876, 2009.

[113] J. Pujol, R. Sangüesa, and J. Delgado. Extracting reputation in multi agent systems by means of social network topology. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 467–474, 2002.

[114] M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In *The Semantic Web - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 351–368. 2003.

[115] L. Roditty and V. Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the 45th Annual ACM Symposium on the Theory of Computing (STOC 2013)*, pages 515–524, 2013.

[116] R. A. Rossi and N. Ahmed. Network repository. accesssed February 28, 2014. `http://networkrepository.com`.

[117] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009.

[118] A. Sala, L. Cao, C. Wilson, R. Zablit, H. Zheng, and B. Zhao. Measurement-calibrated graph models for social network experiments. In *Proceedings of the 19th International Conference on the World Wide Web (WWW 2010)*, pages 861–870, 2010.

[119] R. Sarukkai. Link prediction and path analysis using Markov chains. *Computer Networks*, 33(1):377–386, 2000.

[120] J. Scott. *Social Network Analysis: A Handbook*. Sage Publications, Inc, 1991.

[121] C. Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46(4), article 45, 2014.

[122] L. Šubelj and M. Bajec. Model of complex networks based on citation dynamics. In *Proceedings of the 22nd International Conference on World Wide Web (WWW 2013)*, pages 527–530, 2013.

[123] L. Takac and M. Zabovsky. Data analysis in public social networks. In *International Scientific Conference and International Workshop on Present Day Trends of Innovations, Lomza, Poland*, 6 pages, 2012.

[124] F. W. Takes. Sokoban: Reversed solving. In *Proceedings of 2nd NSVKI Student Conference*, pages 31–36, 2008.

[125] F. W. Takes and W. A. Kosters. Solving Samegame and its chessboard variant. In *Proceedings of 21st Benelux Conference on Artificial Intelligence (BNAIC 2009)*, pages 249–256, 2009.

[126] F. W. Takes and W. A. Kosters. Applying Monte Carlo techniques to the capacitated vehicle routing problem. In *Proceedings of 22th Benelux Conference on Artificial Intelligence (BNAIC 2010)*, 2010.

[127] F. W. Takes and W. A. Kosters. Determining the diameter of small world networks. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM 2011)*, pages 1191–1196, 2011.

[128] F. W. Takes and W. A. Kosters. Identifying prominent actors in online social networks using biased random walks. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)*, pages 215–222, 2011.

[129] F. W. Takes and W. A. Kosters. The difficulty of path traversal in information networks. In *Proceedings of the International Conference on Knowledge Discovery and Information Retrieval (KDIR 2012)*, pages 138–144, 2012.

[130] F. W. Takes and W. A. Kosters. Computing the eccentricity distribution of large graphs. *Algorithms*, 6(1):100–118, 2013.

[131] F. W. Takes and W. A. Kosters. Mining user-generated path traversal patterns in an information network. In *Proceedings of the IEEE/ACM International Conference on Web Intelligence (WI 2013)*, pages 284–289, 2013.

[132] F. W. Takes and W. A. Kosters. Adaptive landmark selection strategies for fast shortest path computation in large real-world graphs. In *Proceedings of the IEEE/ACM International Conference on Web Intelligence (WI 2014)*, pages 27–34, 2014.

[133] J. Teevan, C. Alvarado, M. Ackerman, and D. Karger. The perfect search engine is not enough: A study of orienteering behavior in directed search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 415–422, 2004.

[134] M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005.

[135] K. Thulasiraman and M. N. Swamy. *Graphs: Theory and Algorithms*. John Wiley & Sons, 2011.

[136] K. Tretyakov, A. Armas-Cervantes, L. García-Bañuelos, J. Vilo, and M. Dumas. Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM 2011)*, pages 1785–1794, 2011.

[137] C. Walshaw. The graph partitioning archive. Accessed February 28, 2014. `http://staffweb.cms.gre.ac.uk/~c.walshaw/partition`.

[138] P. Wang, J. Hu, H. Zeng, and Z. Chen. Using Wikipedia knowledge to improve text classification. *Knowledge and Information Systems*, 19(3):265–281, 2009.

[139] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

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

[141] R. West and J. Leskovec. Automatic versus human navigation in information networks. In *Proceedings of the AAAI International Conference on Weblogs and Social Media (ICWSM 2012)*, pages 362–369, 2012.

[142] R. West and J. Leskovec. Human wayfinding in information networks. In *Proceedings of the 21st ACM International World Wide Web Conference (WWW 2012)*, pages 619–628, 2012.

[143] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, 3rd edition, 2011.

[144] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the KDD Workshop on Mining Data Semantics*, article 3, 2012.

[145] J. Yu and J. Cheng. Graph reachability queries: A survey. In *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 181–215. 2010.

[146] J. Yu, J. Thom, and A. Tam. Ontology evaluation using Wikipedia categories for browsing. In *Proceedings of the 16th ACM International Conference on Information and Knowledge Management (CIKM 2007)*, pages 223–232, 2007.

[147] R. Yuster and U. Zwick. Answering distance queries in directed graphs using fast matrix multiplication. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (FOCS 2005)*, pages 389–396, 2005.

[148] J. Zhang, M. Ackerman, and L. Adamic. Expertise networks in online communities: Structure and algorithms. In *Proceedings of the 16th International Conference on World Wide Web (WWW 2007)*, pages 221–230, 2007.

[149] V. Zlatić, M. Božičević, H. Štefančić, and M. Domazet. Wikipedias: Collaborative web-based encyclopedias as complex networks. *Physical Review E*, 74(1):016115, 2006.

[150] U. Zwick. Exact and approximate distances in graphs: A survey. In *Algorithms ESA 2001*, volume 2161 of *Lecture Notes in Computer Science*, pages 33–48. 2001.

Samenvatting

Dit proefschrift gaat over algoritmen voor het analyseren van netwerken, in de informatica vaak *grafen* genoemd. Een netwerk bestaat uit objecten (*knopen*) die met elkaar verbonden zijn door middel van links (*takken*). In tegenstelling tot synthetische grafen die doorgaans het resultaat zijn van het toepassen van een bepaald wiskundig model, ligt de nadruk hier op “real-world” grafen, waarmee wordt bedoeld dat de betreffende graaf is gebaseerd op verzamelde data uit een bestaand domein. Een voorbeeld is een online *sociaal netwerk* zoals Facebook, waarin personen met elkaar zijn verbonden door vriendschappen, of een *webgraaf*, een netwerk waarin internetpagina’s door middel van links naar elkaar verwijzen. Een ander voorbeeld is een netwerk van wetenschappers, waarin de onderlinge relaties tussen de wetenschappers bijvoorbeeld worden bepaald op basis van of zij elkaar citeren of co-auteurs zijn van een artikel (zie bijvoorbeeld Figuur 1.2 in Hoofdstuk 1).

Alhoewel de grafen die in dit proefschrift worden bekeken van elkaar verschillen in termen van wat voor data zij representeren, toont de structuur van deze grafen verrassende overeenkomsten. Zo zijn “real-world” grafen doorgaans “sparse”, wat betekent dat het aantal takken van de graaf klein is ten opzichte van het maximale aantal takken. Desalniettemin bevatten dergelijke grafen vaak één grootste samenhangende component waarin doorgaans het merendeel (meer dan 99%) van de knopen zich bevindt. De distributie van de graad (het aantal buren van een knoop) over alle knopen van de graaf volgt vrijwel altijd een machtsfunctie met een lange staart, wat betekent dat er veel knopen zijn met een relatief lage graad, en enkele knopen met een hele hoge graad ver boven de gemiddelde graad. Deze knopen doen doorgaans dienst als zogenaamde “‘hubs”, die ondanks het feit dat de graaf “sparse” is, ervoor zorgen dat de gemiddelde afstand tussen twee knopen klein is ten opzichte van totale het aantal knopen. Deze zogenaamde “small-world” eigenschap wordt vaak geassocieerd met “six degrees of separation”, een theorie uit de sociologie die zegt dat twee willekeurige personen doorgaans slechts zes handschuddingen van elkaar verwijderd zijn.

Voor informatici ligt de uitdaging op het gebied van de analyse van grafen onder andere in het efficiënt opslaan, zoeken en rekenen in deze grafen met behulp van algoritmen. Hierbij zijn traditionele graafalgoritmen vaak niet praktisch inzetbaar. Zo wordt voor het berekenen van de *afstand* tussen twee knopen traditioneel het kortstepadalgoritme van Dijkstra ingezet, of Breadth First Search wanneer de graaf ongewogen is. Dergelijke algoritmen zijn echter te complex in termen van tijd en ruimte wanneer de graaf uit miljoenen knopen en misschien wel honderden miljoenen of miljarden takken bestaat, en er duizenden kortste paden per seconde berekend dienen te worden. Een veelgebruikte techniek om toch snel de afstand tussen twee knopen in een grote graaf te bepalen maakt gebruik van zogenaamde “landmarks” waarvoor de afstanden vooraf zijn uitgerekend en waarvia vervolgens kan worden genavigeerd wanneer de afstand tussen twee willekeurige knopen berekend dient te worden. In Hoofdstuk 5 worden manieren voor het selecteren van een dergelijke verzameling landmarks besproken, en blijkt dat zowel een gespreide ligging als een hoge centraliteit belangrijk zijn.

De *centraliteit* van een knoop zegt iets over hoe centraal een knoop in de graaf ligt op basis van de structuur van de graaf. De meest simpele maat is “degree centrality”, een maat op basis van de graad van een knoop, waarbij er van uit wordt gegaan dat een knoop centraal ligt wanneer deze veel buren heeft. Deze maat is eenvoudig te berekenen, niet in de laatste plaats omdat grote grafen doorgaans niet als matrix maar als een lijst van knopen en buren (“adjacency list”) worden opgeslagen, waardoor het aantal buren eenvoudig afleesbaar is. Complexere centraliteitsmaten zoals “closeness centrality” en “betweenness centrality” kijken respectievelijk naar de gemiddelde afstand van een knoop tot alle andere knopen en naar het genormaliseerde aantal keren dat een knoop op een kortste pad voorkomt, maar zijn in tegenstelling tot “degree centrality” moeilijker om te berekenen.

In webgrafen is PageRank een veelvoorkomende centraliteitsmaat. Deze techniek geeft een hogere centraliteitswaarde aan een pagina wanneer er een groot aantal andere pagina’s met een hoge centraliteit naar de betreffende pagina verwijzen. Deze maat wordt in de praktijk door zoekmachine Google gebruikt in de vorm van een waarde voor een webpagina tussen 0 (niet belangrijk) en 10 (zeer belangrijk). In Hoofdstuk 6 worden diverse centraliteitsmaten toegepast op de vriendschapsgraaf van een groot Nederlands online sociaal netwerk, en blijkt dat voor het vinden de prominente personen binnen dit netwerk zowel de graad van een persoon als het genormaliseerde aantal driehoeksrelaties van de vrienden van een persoon een rol speelt.

Waar kortste paden en centraliteitsmaten doorgaans een eigenschap van één of enkele knopen berekenen, zijn er ook maten die iets zeggen over de volledige graaf. Een dergelijke maat is de *diameter*: de maximale afstand tussen twee knopen ofwel de lengte van een *langste kortste pad* in de graaf. De diameter kan worden gezien als een worst-case maat van afstand, en zegt bijvoorbeeld iets over hoe informatie zich in het ergste geval verspreidt binnen een netwerk. Een naïeve manier om de diameter te berekenen is door middel van het “All Pairs Shortest Path” algoritme, wat voor ieder paar knopen de onderlinge afstand berekent. De hoogst gevonden waarde is vervolgens de diameter. Deze methode is kwadratisch in het aantal knopen en takken en derhalve niet praktisch inzetbaar in het geval van de grote grafen die in dit proefschrift worden bestudeerd. In Hoofdstuk 2 wordt een nieuw algoritme geïntroduceerd wat in staat is om de diameter van een gegeven graaf veel sneller te bepalen door slim gebruik te maken van onder- en bovengrenzen per knoop en voor de graaf in het geheel. Dit zorgt ervoor dat er niet zoals bij de bovengenoemde methode op basis van het APSP algoritme voor iedere knoop een Breadth First Search berekening uitgevoerd dient te worden, maar dat met slechts enkele tientallen berekeningen en wat “book-keeping” de diameter exact bepaald kan worden.

Een alternatieve manier om de bovengenoemde diameter te definiëren, is door te zeggen dat de diameter gelijk is aan de maximale eccentriciteit over alle knopen. De *eccentriciteit* van een knoop is de lengte van een langste kortste pad van die knoop naar een andere knoop. In Hoofdstuk 3 wordt zowel een exact algoritme als een slimme afschattende methode gepresenteerd om de eccentriciteit van alle knopen in een graaf te berekenen. De bovengenoemde methode kan tevens worden ingezet om andere zogenaamde extreme afstandsmaten te berekenen. Voorbeelden zijn de *straal* (minimale eccentriciteit over alle knopen), het *centrum* (de verzameling knopen met een eccentriciteit gelijk aan de straal) en de *periferie* (de verzameling knopen met een eccentriciteit gelijk aan de diameter) van een graaf (zie Hoofdstuk 4).

Naast de hierboven beschreven algoritmen voor het berekenen van eigenschappen van (de knopen van) een graaf, zijn er voor informatici bij grote grafen ook uitdagingen op het gebied van data mining. *Data mining* heeft veelal als doel om kennis of informatie te verkrijgen uit data, en de gebruikte methoden zijn doorgaans in te delen in voorspellende en beschrijvende technieken.

Voorspellende data miningtechnieken hebben als doel het voorspellen van bepaalde attributen van (groepen van) objecten in de data. In Hoofdstuk 7 wordt gekeken naar een dataset van door gebruikers gegeneerde klikpaden in het informatienetwerk van de online encyclopedie Wikipedia. Daarbij ligt de nadruk op het bepalen van de moeilijkheid voor een gebruiker van het vinden van een bepaalde doelpagina door het klikken op de links die aanwezig zijn in de diverse Wikipedia-artikelen. Een analyse van meer dan twee miljoen klikpaden toont aan dat lokale eigenschappen van de doelpagina voldoende in staat zijn om met hoge precisie te bepalen hoe moeilijk het voor een gebruiker is om een bepaalde pagina te vinden.

Beschrijvende data miningtechnieken proberen doorgaans om de informatie die zich in data bevindt naar boven te krijgen, bijvoorbeeld door te zoeken naar patronen die niet direct zichtbaar zijn door de data handmatig te inspecteren. In Hoofdstuk 8 wordt gekeken naar de individuele pagina's binnen de eerder genoemde verzameling klikpaden in Wikipedia, en wordt gekeken hoe de verzameling van knopen die door gebruikers frequent wordt gebruikt om door het netwerk te navigeren verschilt van een verzameling die is geselecteerd met behulp van eerdergenoemde centraliteitsmaten. Het blijkt dat de door gebruikers geselecteerde verzameling pagina's beter helpt om efficiënt door het netwerk te navigeren, dan een verzameling die geselecteerd is met behulp van een centraliteitsmaat.

De toenemende hoeveelheid data die tegenwoordig wordt gegenereerd kan vaak worden gemodelleerd als een graaf. Voor informatici is er vervolgens een uitdaging weggelegd om deze data efficiënt te analyseren en om er geautomatiseerd informatie en kennis uit te extraheren. Door slim te kijken naar de eigenschappen van de graaf en de complexiteit van diverse methoden en technieken, is het vaak zonder een enorme hoeveelheid brute rekenkracht of gespecialiseerde hardware mogelijk om het gewenste resultaat te bereiken.

Curriculum Vitae

Frank Takes is geboren op donderdag 15 mei 1986 te Leidschendam en groeide op in Zoetermeer, waar hij in 2004 op het Alfrink College zijn VWO-diploma behaalde. In 2008 voltooide Frank zijn bachelor Informatica met minor Bedrijfswetenschappen aan de Universiteit Leiden, gevolgd door zijn master Computer Science cum laude in 2010, tevens in Leiden.

Tijdens zijn studie nam Frank deel aan diverse programmeerwestrijden, waaronder de noordwest Europese regionale wedstrijden in 2008 in Stockholm. Hij was student-assistent bij diverse bachelorvakken en als studentambassadeur betrokken bij de voorlichtingsactiviteiten van zowel de opleiding als de universiteit. Daarnaast nam Frank zitting in achtereenvolgens de opleidingscommissie van de opleiding Informatica, de faculteitsraad van de faculteit Wiskunde en Natuurwetenschappen en de universiteitsraad van de Universiteit Leiden.

Van 2010 tot 2014 voerde Frank onder begeleiding van dr. Walter Kosters promotieonderzoek uit aan het Leiden Institute of Advanced Computer Science (LIACS), het informatica-instituut van de Universiteit Leiden. Frank was tijdens zijn promotietijd betrokken bij de practica van de vakken Programmeermethoden en Kunstmatige Intelligentie, gaf diverse gastcolleges over zijn onderzoek, en verzorgde als onderdeel van een aantal vakken van de opleiding Informatica diverse hoorcolleges. Tijdens zijn promotietijd nam hij zitting in de instituutsraad en de commissie voor de organisatie van gezelligheidsactiviteiten voor medewerkers van het instituut. Vanaf 2007 was Frank tevens part-time actief in het bedrijfsleven als freelance IT-er.

Na zijn promotie streeft Frank in de eerste instantie een part-time wetenschappelijke carrière en een part-time carrière in het bedrijfsleven na.

Dankwoord

Deze pagina is uitsluitend beschikbaar in de gedrukte versie van dit proefschrift.

Publication List

Below is a chronological list of publications by the author up to June 2014.

- F. W. Takes and W. A. Kosters. Adaptive landmark selection strategies for fast shortest path computation in large real-world graphs. In *Proceedings of the IEEE/ACM International Conference on Web Intelligence (WI 2014)*, pages 27–34, 2014

- M. Borassi, P. Crescenzi, M. Habib, W. A. Kosters, A. Marino, and F. W. Takes. On the solvability of the six degrees of Kevin Bacon game — A faster graph diameter and radius computation method. In *Proceedings of the 7th International Conference on Fun with Algorithms (FUN 2014)*, volume 8496 of *Lecture Notes in Computer Science*, pages 52–63. 2014

- G. H. Dal, W. A. Kosters, and F. W. Takes. Fast diameter computation of large sparse graphs using GPUs. In *Proceedings of the 22nd IEEE International Conference on Parallel, Distributed and Network-based Processing (PDP 2014)*, pages 632–639, 2014

- F. W. Takes and W. A. Kosters. Mining user-generated path traversal patterns in an information network. In *Proceedings of the IEEE/ACM International Conference on Web Intelligence (WI 2013)*, pages 284–289, 2013

- P. D. Kusuma, D. Radosavljevik, F. W. Takes, and P. van der Putten. Combining customer attribute and social network mining for prepaid mobile churn prediction. In *Proceedings of 22th Belgian Netherlands Conference on Machine Learning (Benelearn 2013)*, pages 50–58, 2013

- F. W. Takes and W. A. Kosters. Computing the eccentricity distribution of large graphs. *Algorithms*, 6(1):100–118, 2013

- M. Luiten, W. A. Kosters, and F. W. Takes. Topical influence on Twitter: A feature construction approach. In *Proceedings of 24th Benelux Conference on Artificial Intelligence (BNAIC 2012)*, pages 139–146, 2012

- F. W. Takes and W. A. Kosters. The difficulty of path traversal in information networks. In *Proceedings of the International Conference on Knowledge Discovery and Information Retrieval (KDIR 2012)*, pages 138–144, 2012

- F. W. Takes and W. A. Kosters. Determining the diameter of small world networks. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM 2011)*, pages 1191–1196, 2011

- F. W. Takes and W. A. Kosters. Identifying prominent actors in online social networks using biased random walks. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)*, pages 215–222, 2011

- A. M. Kentsch, W. A. Kosters, P. van der Putten, and F. W. Takes. Exploratory recommendations using Wikipedia's linking structure. In *Proceedings of 20th Belgium Netherlands Conference on Machine Learning (Benelearn 2011)*, pages 61–68, 2011

- F. W. Takes and W. A. Kosters. Applying Monte Carlo techniques to the capacitated vehicle routing problem. In *Proceedings of 22th Benelux Conference on Artificial Intelligence (BNAIC 2010)*, 2010 (based on the author's master thesis)

- F. W. Takes and W. A. Kosters. Solving Samegame and its chessboard variant. In *Proceedings of 21st Benelux Conference on Artificial Intelligence (BNAIC 2009)*, pages 249–256, 2009 (based on the author's master project study)

- F. W. Takes. Sokoban: Reversed solving. In *Proceedings of 2nd NSVKI Student Conference*, pages 31–36, 2008 (based on the author's bachelor thesis)

Titles in the IPA Dissertation Series since 2008

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates*. Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathematics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calamé. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems*. Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification*. Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. Khadim. *Process Algebras for Hybrid Systems: Comparison and Development*. Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. Markovski. *Real and Stochastic Time in Process Algebras for Performance Evaluation*. Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. Kastenberg. *Graph-Based Software Specification and Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. Buhan. *Cryptographic Keys from Noisy Data Theory and Applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. Marin-Perianu. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean*. Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques*. Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming Ready for Prime Time*. Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Transformation*. Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML*. Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. Czenko. *TuLiP - Reshaping Trust Management*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. Chen. *Clocks, Dice and Processes*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web*. Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory*. Faculty of Science, Mathematics and Computer Science, RU. 2009-19

B. Ploeger. *Improved Verification Methods for Concurrent Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-20

T. Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

R. Li. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis*. Faculty of Mathematics and Natural Sciences, UL. 2009-22

J.H.P. Kwisthout. *The Computational Complexity of Probabilistic Networks*. Faculty of Science, UU. 2009-23

T.K. Cocx. *Algorithmic Tools for Data-Oriented Law Enforcement*. Faculty of Mathematics and Natural Sciences, UL. 2009-24

A.I. Baars. *Embedded Compilers*. Faculty of Science, UU. 2009-25

M.A.C. Dekker. *Flexible Access Control for Dynamic Collaborative Environments*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

J.F.J. Laros. *Metrics and Visualisation for Crime Analysis and Genomics*. Faculty of Mathematics and Natural Sciences, UL. 2009-27

C.J. Boogerd. *Focusing Automatic Code Inspections*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

M.R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

J. Endrullis. *Termination and Productivity*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

T. Staijen. *Graph-Based Specification and Verification for Aspect-Oriented Languages*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

Y. Wang. *Epistemic Modelling and Protocol Dynamics*. Faculty of Science, UvA. 2010-05

J.K. Berendsen. *Abstraction, Prices and Probability in Model Checking Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2010-06

A. Nugroho. *The Effects of UML Modeling on the Quality of Software*. Faculty of Mathematics and Natural Sciences, UL. 2010-07

A. Silva. *Kleene Coalgebra*. Faculty of Science, Mathematics and Computer Science, RU. 2010-08

J.S. de Bruin. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications*. Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. Costa. *Formal Models for Component Connectors*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. Jaghoori. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services*. Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. Bakhshi. *Gossiping Models: Formal Analysis of Epidemic Protocols*. Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. Arnoldus. *An Illumination of the Template Enigma: Software Code Generation with Templates*. Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. Zambon. *Towards Optimal IT Availability Planning: Methods and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

L. Astefanoaei. *An Executable Theory of Multi-Agent Systems Refinement*. Faculty of Mathematics and Natural Sciences, UL. 2011-04

J. Proença. *Synchronous coordination of distributed components*. Faculty of Mathematics and Natural Sciences, UL. 2011-05

A. Moralı. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

M. van der Bijl. *On changing models in Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

C. Krause. *Reconfigurable Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-08

M.E. Andrés. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2011-09

M. Atif. *Formal Modeling and Verification of Distributed Failure Detectors*. Faculty of Mathematics and Computer Science, TU/e. 2011-10

P.J.A. van Tilburg. *From Computability to Executability – A process-theoretic view on automata theory*. Faculty of Mathematics and Computer Science, TU/e. 2011-11

Z. Protic. *Configuration management for models: Generic methods for model comparison and model co-evolution*. Faculty of Mathematics and Computer Science, TU/e. 2011-12

S. Georgievska. *Probability and Hiding in Concurrent Processes*. Faculty of Mathematics and Computer Science, TU/e. 2011-13

S. Malakuti. *Event Composition Model: Achieving Naturalness in Runtime Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

M. Raffelsieper. *Cell Libraries and Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-15

C.P. Tsirogiannis. *Analysis of Flow and Visibility on Triangulated Terrains*. Faculty of Mathematics and Computer Science, TU/e. 2011-16

Y.-J. Moon. *Stochastic Models for Quality of Service of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-17

R. Middelkoop. *Capturing and Exploiting Abstract Views of States in OO Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-18

M.F. van Amstel. *Assessing and Improving the Quality of Model Transformations*. Faculty of Mathematics and Computer Science, TU/e. 2011-19

A.N. Tamalet. *Towards Correct Programs in Practice*. Faculty of Science, Mathematics and Computer Science, RU. 2011-20

H.J.S. Basten. *Ambiguity Detection for Programming Language Grammars*. Faculty of Science, UvA. 2011-21

M. Izadi. *Model Checking of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-22

L.C.L. Kats. *Building Blocks for Language Workbenches*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2011-23

S. Kemper. *Modelling and Analysis of Real-Time Coordination Patterns*. Faculty of Mathematics and Natural Sciences, UL. 2011-24

J. Wang. *Spiking Neural P Systems*. Faculty of Mathematics and Natural Sciences, UL. 2011-25

A. Khosravi. *Optimal Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2012-01

A. Middelkoop. *Inference of Program Properties with Attribute Grammars, Revisited*. Faculty of Science, UU. 2012-02

Z. Hemel. *Methods and Techniques for the Design and Implementation of Domain-Specific Languages*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03

T. Dimkov. *Alignment of Organizational Security Policies: Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

S. Sedghi. *Towards Provably Secure Efficiently Searchable Encryption*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

F. Heidarian Dehkordi. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference*. Faculty of Science, Mathematics and Computer Science, RU. 2012-06

K. Verbeek. *Algorithms for Cartographic Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2012-07

D.E. Nadales Agut. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation*. Faculty of Mechanical Engineering, TU/e. 2012-08

H. Rahmani. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2012-09

S.D. Vermolen. *Software Language Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

L.J.P. Engelen. *From Napkin Sketches to Reliable Software*. Faculty of Mathematics and Computer Science, TU/e. 2012-11

F.P.M. Stappers. *Bridging Formal Models – An Engineering Perspective*. Faculty of Mathematics and Computer Science, TU/e. 2012-12

W. Heijstek. *Software Architecture Design in Global and Model-Centric Software Development*. Faculty of Mathematics and Natural Sciences, UL. 2012-13

C. Kop. *Higher Order Termination*. Faculty of Sciences, Department of Computer Science, VUA. 2012-14

A. Osaiweran. *Formal Development of Control Software in the Medical Systems Domain*. Faculty of Mathematics and Computer Science, TU/e. 2012-15

W. Kuijper. *Compositional Synthesis of Safety Controllers*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16

H. Beohar. *Refinement of Communication and States in Models of Embedded Systems*. Faculty of Mathematics and Computer Science, TU/e. 2013-01

G. Igna. *Performance Analysis of Real-Time Task Systems using Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2013-02

E. Zambon. *Abstract Graph Transformation – Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

B. Lijnse. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2013-04

G.T. de Koning Gans. *Outsmarting Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2013-05

M.S. Greiler. *Test Suite Comprehension for Modular and Dynamic Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-06

L.E. Mamane. *Interactive mathematical documents: creation and presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2013-07

M.M.H.P. van den Heuvel. *Composition and synchronization of real-time components upon one processor*. Faculty of Mathematics and Computer Science, TU/e. 2013-08

J. Businge. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins*. Faculty of Mathematics and Computer Science, TU/e. 2013-09

S. van der Burg. *A Reference Architecture for Distributed Software Deployment*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-10

J.J.A. Keiren. *Advanced Reduction Techniques for Model Checking*. Faculty of Mathematics and Computer Science, TU/e. 2013-11

D.H.P. Gerrits. *Pushing and Pulling: Computing push plans for disk-shaped robots, and dynamic labelings for moving points*. Faculty of Mathematics and Computer Science, TU/e. 2013-12

M. Timmer. *Efficient Modelling, Generation and Analysis of Markov Automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-13

M.J.M. Roeloffzen. *Kinetic Data Structures in the Black-Box Model*. Faculty of Mathematics and Computer Science, TU/e. 2013-14

L. Lensink. *Applying Formal Methods in Software Development*. Faculty of Science, Mathematics and Computer Science, RU. 2013-15

C. Tankink. *Documentation and Formal Mathematics — Web Technology meets Proof Assistants*. Faculty of Science, Mathematics and Computer Science, RU. 2013-16

C. de Gouw. *Combining Monitoring with Run-time Assertion Checking*. Faculty of Mathematics and Natural Sciences, UL. 2013-17

J. van den Bos. *Gathering Evidence: Model-Driven Software Engineering in Automated Digital Forensics*. Faculty of Science, UvA. 2014-01

D. Hadziosmanovic. *The Process Matters: Cyber Security in Industrial Control Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-02

A.J.P. Jeckmans. *Cryptographically-Enhanced Privacy for Recommender Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-03

C.-P. Bezemer. *Performance Optimization of Multi-Tenant Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2014-04

T.M. Ngo. *Qualitative and Quantitative Information Flow Analysis for Multi-threaded Programs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-05

A.W. Laarman. *Scalable Multi-Core Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-06

J. Winter. *Coalgebraic Characterizations of Automata-Theoretic Classes*. Faculty of Science, Mathematics and Computer Science, RU. 2014-07

W. Meulemans. *Similarity Measures and Algorithms for Cartographic Schematization*. Faculty of Mathematics and Computer Science, TU/e. 2014-08

A.F.E. Belinfante. *JTorX: Exploring Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-09

A.P. van der Meer. *Domain Specific Languages and their Type Systems*. Faculty of Mathematics and Computer Science, TU/e. 2014-10

B.N. Vasilescu. *Social Aspects of Collaboration in Online Software Communities*. Faculty of Mathematics and Computer Science, TU/e. 2014-11

F.D. Aarts. *Tomte: Bridging the Gap between Active Learning and Real-World Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2014-12

N. Noroozi. *Improving Input-Output Conformance Testing Theories*. Faculty of Mathematics and Computer Science, TU/e. 2014-13

M. Helvensteijn. *Abstract Delta Modeling: Software Product Lines and Beyond*. Faculty of Mathematics and Natural Sciences, UL. 2014-14

P. Vullers. *Efficient Implementations of Attribute-based Credentials on Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2014-15

F.W. Takes. *Algorithms for Analyzing and Mining Real-World Graphs*. Faculty of Mathematics and Natural Sciences, UL. 2014-16