TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Genomorientierte Bioinformatik

Next Generation Knowledge Extraction from Biomedical Literature with Semantic Big Data Approaches

Benedikt N. X. Wachinger

Vollständiger Abdruck der von der Fakultät Wissenschaftszentrum Weihenstephan für Ernährung, Landnutzung und Umwelt der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. D. Frischman
Prüfer der Dissertation:	1. Univ.-Prof. Dr. H.-W. Mewes 2. Priv.-Doz. Dr. R. Küffner, Ludwig-Maximilians-Universität München

Die Dissertation wurde am 07.01.2013 bei der Technischen Universität München eingereicht und durch die Fakultät Wissenschaftszentrum Weihenstephan für Ernährung, Landnutzung und Umwelt am 25.04.2013 angenommen.

Abstract

The last couple of years have been characterized by a massive increase in publication volume indexed in Pubmed and Pubmed Central. The freely accessible part of both resources contains more than 22 million documents in the beginning of 2012 which increases with a speed of two to three new entries every minute of the year. This exponentially increasing amount of published research articles makes it obvious that most of the knowledge contained in those articles is lost within the sea of innumerable publications because no useful knowledge management system is able to deal with those amounts of information in a time-efficient manner. For this reason, multiple text mining systems have been developed with varying degrees of useability, efficiency, and further applicability that attempt to automatically extract relevant knowledge from texts written by humans.

The text mining system, Excerbt, was developed at the Institute of Bioinformatics and Systems Biology at the Helmholtz Zentrum München. While providing a proven solution to the problem of extracting relevant knowledge from texts by the use of a technique called semantic role labeling, it lacked efficient features in dealing with exponentially growing big data amounts of publications.

Within the scope of this thesis, I brought the text mining system Excerbt to a new level by combining its proven approaches with a completely new cloud-based hard- and software architecture and an according redevelopment of the core processes. Specifically, the new system now employs an approach based on the Hadoop MapReduce framework and the bigtable database HBase. This has several advantages regarding efficiency of knowledge extraction as well as fast ad-hoc analyses on extracted data with enormously reduced manual work.

The extracted data of the new Excerbt is now stored as a topic map in a graphical representation in HBase. The graph's nodes and edges represent biomedical entities and their interactions as derived from literature. In the beginning of 2012, this graph contains approximately 600.000 nodes and 75 million edges. Results of ad-hoc analyses on this graph indicate that it exhibits the scale-free property as expected and often observed for organically grown networks.

Based on the knowledge stored in Excerbt I have further developed a method that is able to point scientists to under- and over-specified areas within a graph that are suggested for further research with a scoring system. These areas denote edges between nodes that either exist in the graph and are unlikely or do not exist but seem plausible because of the neighboring network structure. The approach is loosely based on locally clustered egocentric network analysis and assigns scores to all existing and hypothetical edges in a network based on their surroundings.

With these developments I have brought knowledge extraction to the next generation. The capability of the system of handling exponentially increasing data sets together with automated suggestion of research targets covers a broad range of knowledge extraction and management and opens up innumerable ways to perform further research automatically extracted big data knowledge.

Zusammenfassung

Die letzten Jahre waren von einem massiven Anstieg an Publikationsvolumen, das in Pubmed und Pubmed Central indiziert ist, geprägt. Der frei zugängliche Teil von beiden Ressourcen enthält seit Anfang Jahr 2012 mehr als 22 Millionen Einträge, die mit einer Geschwindigkeit von zwei bis drei neuen Artikeln pro Minute wachsen.

Die Anzahl an veröffentlichten wissenschaftlichen Artikeln innerhalb jedes beliebigen wissenschaftlichen Feldes macht es offensichtlich, dass das meiste Wissen innerhalb dieser Artikel im Sumpf der unzählbaren Publikationen verschwindet. Um diese Größenordnungen an Publikationen sinnvoll verwalten zu können, bedarf es leistungsfähiger Methoden der Wissensextraktion, die zum einen in der Lage sind relevantes Wissen aus Texten, die von Menschen geschrieben wurden, zu extrahieren und zum anderen exponentiell wachsende Big Data Mengen zu verarbeiten.

Im Rahmen dieser Arbeit habe ich das Text Mining System Excerbt auf eine neue Ebene gebracht, indem seine bereits bewährten Ansetze mit vollständig neuen, Cloud-basierten Hard- und Softwareansätzen und einer entsprechenden Neuentwicklung der Core-Prozesse gekoppelt wurden. Das neuartige System setzt nun spezifisch einen Ansatz ein, der auf dem Framework Hadoop MapReduce und der Bigtable Datenbank HBase basiert. Das System extrahiert Wissen aus Text mittels eines semantischen Ansatzes um Relationen zwischen biomedizinischen Entitäten innerhalb einzelner Sätze zu detektieren. Dieses extrahierte Wissen wird anschließend in der Form eines riesigen Graphen mit derzeit etwa 600.000 Knoten und mehr als 75 Millionen Kanten abgespeichert.

Diese Dissertation zeigt die Kompliziertheit und die Schwierigkeiten die bei der Implementation eines solchen Systems aufkommen. Sie zeigt außerdem Big Data Methoden, die spezifisch dafür entwickelt wurden, um mit den massiven Graphdaten zu arbeiten und um großräumige Netzwerkanalysen durchzuführen. Ergebnisse basierend auf dem Excerbt Literaturnetzwerk zeigen, dass der Graph die Eigenschaft der Skaleninvarianz beinhaltet. Weitere Ergebnisse zeigen, dass die beschriebenen Ansätze in Verbindung mit Big Data Analyse funktionieren und ein effizientes Verarbeiten der Schritte der Prozessierungspipeline wie auch schnelle ad-hoc Abfragen an die riesigen Datenmengen mittlerweile zur Realität geworden sind.

Basierend auf dem Wissen, das in Excerbt gespeichert ist, habe ich weiterhin eine Methode entwickelt, die in der Lage ist, Wissenschaftler auf über- und unter-spezifizierte Bereiche in Netzwerken hinzuweisen, die es wert sind weiter untersucht zu werden. Der Ansatz basiert grob auf lokal geclusterten egozentrischen Netzwerken und markiert existierende und nicht-existierende Kanten im Netzwerk mit spezifischen Kennzahlen basierend auf der unmittelbaren Umgebung innerhalb des Graphen.

Mit diesen Entwicklungen habe ich das Gebiet der Wissensextraktion in die nächste Generation gebracht. Die Möglichkeiten des Systems, exponentiell wachsende Datenmengen zu verarbeiten und die

automatisierte Identifikation potentieller Forschungsziele decken ein vollständiges Wissensmanagementsystem ab und öffnen unzählbare Möglichkeiten für neue Forschung auf Text- und Graphdaten.

Acknowledgements

Writing this doctoral thesis would not have been possible without the help and support of the people around me, to only some of whom it is possible to give particular mention here.

First and foremost, I would like to express my deepest gratitude to Dr. Volker Stümpflen. Without him, it would not have been possible to write this thesis. As my supervisor he inspired me with his ideas and supported me with his invaluable advice and constructive discussions. His door was always open whenever I had questions or doubts and he was able to guide me back on track whenever research seemed to take a wrong turn.

I would also like to cordially thank Prof. Hans-Werner Mewes who gave me the opportunity to do my doctorate at the Institute of Bioinformatics and Systems Biology. Also, I am thankful for his supporting input, his confidence in me and my work, and his door that was also always open whenever needed. Likewise I want to thank him for the possibility of acquiring a compute cluster without which this work would not have been possible.

At this place, I also want to thank Dr. Robert Küffner, who, as part of the thesis and examination committees, provided me with useful advice on further developments of my work and during the writing process.

Further particular thanks go to my long-term student assistant, Ulrich Köhler, who helped me implement some of the more daunting tasks of the software developments. I would also like to thank Sebastian Ullherr, without whom Excerbt's web interface would not exist in its current form. Additional thanks go to my master's and bachelor's students Barbara Hummel, Nikolas Gross, Robert Strache, and Syeda Tanzeem H. Charu. I very much appreciated the nice working environment, which my colleagues Michael Greeff, Philipp Blohm, Mara Hartsperger, Matthias Walter, Sebastian Vosberg and all other co-workers from the institute had a substantial influence on.

Last, but by no means least, I thank my friends and family for their enduring support and encouragement throughout, especially in the last stage of writing this thesis.

Table of Contents

1 Introduction

1.1 Motivation and Objectives

The last couple of years have been characterized by a voracious increase in data volume in all areas of research (biomedical and other) and regular life. Starting with scientific experiments that produce more and more data, such as the Large Hadron Collider project, through to social media networks, such as Facebook, that have an alarming data increase rate (Constine, 2012).

A recent study by the IDC (Gantz and Reinsel, 2011) predicted a 50-fold increase in total data volume world-wide for the next decade. While the sum of biomedical data is a comparatively small part of it, the growth rates can be expected to be at similar magnitudes. NCBI's Next Generation Sequencing read archive, for example, has currently a total volume of approximately 1.7 PetaBytes of data[1]. Pubmed MEDLINE has already indexed nearly 22 million citations and continues to grow with a double-exponential increase (Hunter and Cohen, 2006; Krallinger et al., 2008). A recent survey of the number of newly added citations to Pubmed MEDLINE shows a significant increase over the last years, already exceeding 50.000 and more new entries each month (Figure 1.1) which means nearly two to three new publications every minute, every day of the year. The times where anybody could read and let alone comprehend all available literature on any particular subject have long been gone. The trend to bigger and exponentially increasing data volumes, a multitude of different data sources and an increased desire to be able to process all that in as little time as possible can basically be summarized into one question: What would it take to get knowledge out of data at those magnitudes?

For Systems Biology, the availability of large-scale data in computer-readable, structured formats is also of vital importance. The modeling of systems is a large part in understanding the etiology of complex diseases and other processes in living organisms. Denis Noble defined Systems Biology as "putting together rather than taking apart, integration rather than reduction" (Noble, 2008), which clearly corresponds to a scenario where making use of all available resources containing knowledge for common approaches is the case instead of taking a reductionist approach and using those resources only in a non-integrated fashion. Put differently: In order to fully comply to the Systems Biology paradigm we try to adhere, methods are needed that structure the unstructured (i.e. biomedical publications) in order to enable an integration via computational methods.

Several methods already deal with necessary groundwork of computer hardware able to process those kinds of data (Trelles et al., 2011; van Beek, 2006). Developments have, however, shown that the amount of knowledge extracted from data stored in resources accessible by automated methods is still limited. Most new findings obtained by analyzing data (such as large-scale experimental data,

[1] http://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=download$_$reads, accessed 2012-05

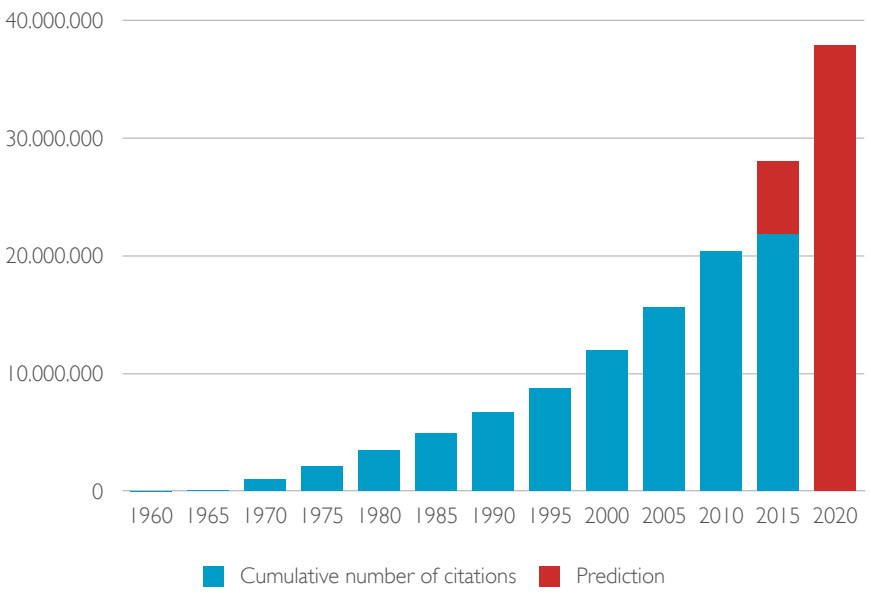


Figure 1.1: Cumulative number of citations in Pubmed MEDLINE per year. Clearly, an exponential increase in data volume in the number of citations of Pubmed MEDLINE can be seen. Counted values are blue, predicted values are red. The value at 2015 contains counted values until May 2012 and estimates for the remainder. Estimates show that the number of citations in Pubmed will increase to 38 million citations by the year 2020. Data as at May 2012.

high-throughput data) are written down in an unstructured way in text publications of any kind. These are straightforward for humans but difficult for computers to understand and process.

Recent developments in computer science introduced the concept of cloud computing a couple of years ago. Within this concept, computing power can be provided as a resource that can be acquired at any given time in any amount possible. Because of this, the development of truly scalable computer systems, that are in a position to cope with arbitrarily large amounts of data, is now a reality.

Methods that profoundly analyze available literature in order to structure the unstructured information, in other words to extract specific assertions about biomedical entities, are called text mining systems and have been around for some years now in one form or the other. They usually employ specific data mining techniques to extract knowledge of a certain kind, such as protein phosphorylation sites, or generalize to common concepts between an unlimited amount of biomedical knowledge types. The text mining system Excerbt (Barnickel, 2009) has been invented and developed by a former colleague at my institute, the Institute of Bioinformatics and Systems Biology at the Helmholtz Zentrum München. All text mining systems in common, however, is that they are unable to process exponentially increasing big data in an efficient way. The rise in publication volume presents such big data and could not be effectively handled by Excerbt.

The question from before, however, has still no answer yet. One goal within the scope of this thesis was therefore to transfer the approaches that have been used throughout the general industry (such as cloud computing technologies), but not (yet) been applied in biology, to biomedical research, in particular

the semantic knowledge extraction from literature. The Excerbt text mining system was used as a reference for implementing the knowledge extraction tasks. Due to some evident architectural changes, the algorithms to some of the processes of the extraction pipeline in Excerbt had to be completely revised in order to allow for an efficient use of cloud computing concepts. In particular, the example with the most complete rework needed was the named entity recognition, the task of identifying names of biomedical entities within texts, from a dictionary-based perspective. Several new techniques have been developed within the scope of this thesis to be able to perform this task better and more efficient for exponentially increasing data volumes.

This thesis will make clear that the transferral of big data technologies onto the biomedical domain will open up numerous new possibilities for analysis of data and subsequent knowledge inferences that could not have been performed before. For example, an analysis of word and token frequencies in biomedical texts allows an estimate of the size of the biomedical domain language in a very short time.

When dealing with big data volumes, the need for fully automated methods arises. Knowledge extracted via text mining methods can be stored in several ways, the dominant technique being a representation as graphs. Naturally, with large input data volumes, the output data volume will rise too. Handling a graph with several million nodes and an order of magnitude more edges between those nodes is impossible without specialized techniques. Another major goal of this thesis was therefore to determine, how this network of knowledge can be used in an automated fashion in order to provide insights into the nature of particular biomedical processes that can only be seen when taking a holistic network approach into consideration. Put differently, the development of methods to tackle big graph problems in an automated fashion then has the ability to provide scientists with fine-grained insights but without losing the big picture.

In sum, this thesis presents the following: First of all, a newly developed version of a text mining system that is now capable of handling big data volumes. Based upon the processes involved in the Excerbt text mining system, I will introduce several methods that had to be completely changed in order to be able to achieve a high scalability. An in-depth analysis of the data generated by these approaches, several new opportunities for data analysis have been opened up. Second of all, the data created with Excerbt is of network structure. This structure has been used to implement automated methods that consider the integral network in order to provide detailed insights. Specifically, the method attempts to identify gaps within the network that could be of reasonable interest to researchers.

1.2 Outline

This thesis is divided into seven chapters. This first Chapter 1 introduces and motivates the work and gives an outline of the remaining six chapters with a summarized description of their contents. Each chapter usually begins with a short introduction or motivation and ends with a summary of the discussed concepts. In the last Chapter 7, a conclusion of the work and an outlook into future possibilities is given.

Chapter 2, the preliminaries, introduces the reader to the basic mathematical, biological, and technological concepts that are used throughout this thesis. It begins with an introduction into network

theory and a discussion of the most important terms and concepts that are necessary to understand the algorithms that were developed in later chapters. After that, an introduction into cloud computing and applied technologies is given, particularly into the MapReduce framework in general and with Hadoop in particular. The chapter also introduces a recently invented category of database types, NoSQL databases. Here, a representative selection of systems is presented in order to familiarize the reader with the concept. Specifically, the HBase database will be used throughout the remainder of the thesis. Furthermore, I will put the matter of natural language processing techniques in a nutshell and describe some widespread techniques that are used throughout the industry. Last but not least, I will cover the subject of knowledge representation by topic maps in combination with subject-centric computing.

In Chapter 3 I will then introduce the general ideas behind text mining. I will give an overview of the different text mining types, specifically co-occurrence, relation-based, statistical, and sentiment analyzing text mining systems. This chapter also includes a section about related work in the field of text mining and I will describe some general guidelines on how to successfully develop a text mining system such as Excerbt as presented in this thesis. Additionally, I will discuss some of the main difficulties when working with such systems.

Chapter 4 then presents the major technological work that was done in the scope of this thesis. It describes, how the concepts developed in the previous chapter can be deployed to work with big data and discuss the most relevant differences that arise for large-volume data handling techniques in contrast to regular methods. In principle, this chapter describes all the relevant concepts and necessary transformations in order to build a working text mining system that can cope with the massive volumes of exponentially increasing data volumes and how to turn them into a useable big knowledge representation.

Throughout Chapter 5 I will then extensively analyze the data created by the methods of Chapter 4. This chapter gives the most relevant findings when working with big data and illuminates the possibilities when working with big data technologies on existing data. The data created by Excerbt is represented as a graph. As such, I will also cover big graph analysis methods that could not have been done without the techniques developed in previous chapters. Concretely, I will present a broad network analysis of the knowledge graph in Excerbt, with particular mention of degree and clustering coefficient distributions. This chapter will also introduce concepts on how to work with statistical methods on the data corpus (such as the N-Gram approach) in order to gain novel insights into the general composition of biomedical literature. The chapter will also include data- and user-driven evaluations and discuss why the former methods result in inferior performances compared to the latter ones. At last, I will present some applications that have been conducted in collaboration with biologists, where the use of the Excerbt text mining system benefitted their research.

Finally, Chapter 6 takes up on all the previously discussed approaches of text mining and big graph processing in order to develop a specific large-scale application, called HypoGen. It was designed to identify structures in networks that seem unlikely based on the overall network architecture. More specifically, it shows how the structure of a network can be used in order to find edges that seem interesting and introduces a new, network-based approach to handling large graphs for assessing completeness situations within specific areas of the network in order to find under- and over-specified areas.

2 Preliminaries

This chapter's aim is to introduce the reader to the fundamental biological principles and the technological prerequisites that are necessary to fully comprehend this thesis. Starting with a general introduction to biological networks (Section 2.1) together with an overview of network clustering and community detection methods (Section 2.1.2), I will then introduce the various technological foundations. This includes an overview of new, sophisticated big data handling solutions, such as the MapReduce programming model (Section 2.3) and NoSQL data stores (Section 2.4). Since the tools presented in this thesis heavily rely on the use of text mining algorithms, I will give an extensive overview of natural language processing methods in Section 2.5. Lastly, Section 2.6 will introduce necessary concepts on associative computing and the knowledge management concept of Topic Maps.

The topics in this chapter are in no particular order except that later sections assume a familiarity with previous sections. When appropriate, I will reference to later sections of this work.

2.1 Network Basics

In the biomedical research area, especially in Systems Biology, networks are versatile tools that can be used to analyze the complex interactions of the players of biological systems. Examples for networks include gene-regulatory networks or metabolic networks, but also in a broader sense collaboration networks, co-authorship networks or — in case of non-biological networks — friendship networks.

Throughout the last century, biology basically was a science of single molecules. Single proteins, for example, were identified and then every interaction that this particular protein could perform was experimentally verified. The rise of Systems Biology at around the year 2000 and the subsequent enablement of large-scale methods lead to an abundance of genomic data and tools which made the single molecule approach less favorable because of the now existing possibilities. Barabási and Oltvai (2004) wrote in their paper

> "There is a clear need to understand how these molecules and the interactions between them determine the function of this enormously complex machinery [...]"

which basically demonstrates why networks gained such popularity especially in the biomedical sciences. As we will see later, networks are a versatile tool to represent knowledge of any kind despite some of their shortcomings, that mostly include intractable algorithms on a computer. Network analysis proved to be a flexible tool to gain new insights into the structure of relationships between biomedical concepts and the function of biological systems (Proulx et al., 2005).

Networks, or more formally *graphs*, are a representation of objects (called *nodes* or *vertices*) that are linked to each other through *edges* or *links*. Nodes in a graph are usually depicted as dots or circles with vertices being displayed as lines connecting two dots at a time. Graphs, where vertices can connect more than two dots at any given time are called *hypergraphs*.

Mathematically, a graph G is a tupel (V, E), where V is a set of vertices and E a set of edges, which are subsets of V. Graphs can be coarsely differentiated into *directed* and *undirected* graphs (see Figure 2.1). In undirected graphs, the edge subsets of V are unordered sets, meaning for any set of two vertices $(a, b) = (b, a)$ holds true, whereas in directed graphs those sets are ordered and therefore not identical. In a directed edge (a, b), a is called the tail or *source* and b is called the head or *target*, meaning the link is directed from a to b.

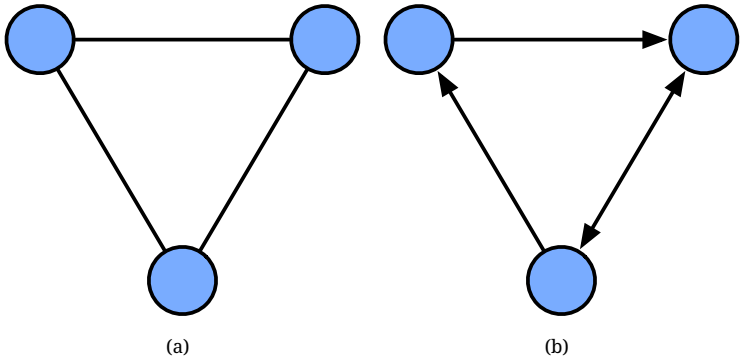


Figure 2.1: An undirected (a) and a directed (b) graph.

In each of the example networks mentioned earlier, nodes describe some relevant concepts (genes, metabolites, people, ...) and edges basically characterize how those concepts communicate or interact with each other.

Nodes that are directly linked to each other are called *neighbors*. The neighborhood network of a particular node n is called the *egocentric network of node n*. An egocentric network of depth 1 includes the node and the direct neighborhood. A network that includes the neighbors' neighbors of node n is in turn called the egocentric network of depth 2 and so on.

The *degree d* of a node n, denoted as $d(n)$, gives the number of edges a node is involved in, if we allow multiple edges between two nodes. Alternatively, if we only allow a single edge between any two nodes, it gives the number of neighbors. In directed networks, the degree can be differentiated into an *incoming degree* (number of edges pointing towards that node, i.e. the node is the target) and *outgoing degree* (number of edges pointing away from that node, i.e. the node is the source). A node with a degree of 0, i.e. the node has no neighbors, is called a *singleton*.

A graph can be traversed along an alternating sequence of nodes and edges, starting from a node and resulting in a node, which is called a *path*. If such a path returns to the original node, we have a circular path. The number of intermediate edges between two nodes in a path is called the *distance*. According to this definition, the distance to directly connected neighbors is therefore one. In directed networks, a graph can only be traversed from sources to gargets, in undirected networks in either direction. If

there is no path in a graph that has the same node twice, as a source *and* as a target, we call the graph *acyclic*. Otherwise it is called a *cyclic* or *circular* graph.

For any two nodes with multiple paths inbetween, the path with the smallest distance is called the *shortest path*. The length of the longest shortest path of all paths in a graph is called the *diameter* of the graph.

A circular path consisting of exactly three nodes and three edges is called a *triangle*. In other words, starting from a node n, a triangle is formed between any two neighbors of node n that have an edge between them. In the undirected network case, the traversal can follow in any direction; in the directed network case, the traversal is usually from source to target node only.

A graph's global properties can be characterized by a multitude of measurements. One famous measurement is the so-called *clustering coefficient* (CC). The CC of a node n gives a measure of how tightly interconnected this particular node is with its neighborhood. It is defined as

$$c(n) = \frac{2l}{k(k-1)} \tag{2.1}$$

where l is the number of links connecting the k neighbors of node n with each other (Barabási and Oltvai, 2004). Congruently, the clustering coefficient can also be defined by using triangles that are formed with node n. Colloquially, $c(n)$ can be computed by:

$$c(n) = \frac{\text{number of observed triangles}}{\text{number of possible triangles}}. \tag{2.2}$$

In order to characterize a full graph, the clustering coefficient of every node is computed and averaged giving an approximation of the interconnectedness in the graph. Generally speaking, the higher this coefficient, the more edges are in the graph.

Cliques are sets of nodes where each of the nodes is connected to every other node in the set (Weisstein, 1999). If we relax this restriction to a definition where most of the possible edges within the set are present, we call it *trusses*. The exact relaxation of this definition usually depends on the application. One could, for example, define that a truss is valid for a specific high percentage of existing edges out of possible edges between those nodes, thereby measuring the completeness of the clique and define a truss based on some thresholds.

A network's structure can be either seen as an *explicit network* where the actual nodes and edges are considered or as an *implicit network* where not the nodes and edges directly but some shared properties are used. Put differently, an implicit network is a projectable version of something else. For example, a network of people and their friends (explicit) could be projected onto an network of people exchanging messages, such as emails (implicit)[2].

2.1.1 Network models

Networks can be broadly classified into different categories, called models. This categorization depends on the overall structure of the network, the probability distributions of node-degrees and their intercon-

[2] Obviously, this only works if we allow only messages between friends, but this suffices as an example here.

nectedness. According to Barabási and Oltvai (2004), three models are of notable importance for biology and have been proven crucial for the understanding of biological networks, such as protein-protein interaction networks or gene-disease networks.

The most basic network model describes a *random network*. There are different mechanisms to produce random networks, the most famous being the *Erdős–Rényi model*. The basic characteristic of that model is that the probability for an edge to connect two particular nodes is equal for any node. The degree distribution therefore follows the Poisson distribution, meaning that most nodes have a degree near the average degree of all nodes (Figure 2.2A).

In the *scale-free networks* model the probability for an edge to be attached to a particular node increases with the number of edges the node already has (also called *preferential attachment*). This favors the creation of a few so-called *hubs*, nodes with a significantly higher degree than average, and many nodes having only a single neighbor (i.e. $d = 1$) (Figure 2.2B). These networks have an interesting characteristic in that they are especially robust against random node attacks. If random nodes are eliminated from the network, the overall structure stays intact. In biology, nodes are often genes and an attack could be a mutation in a gene, leading to a loss of function for that specific gene. Therefore the gene cannot fulfill its task anymore and is eliminated from the network. This happens randomly and because of the nature of biological networks, which are most often scale-free, the overall system that the network describes is robust against those attacks and can still function normally (Albert et al., 2000; Greenbury et al., 2010). The scale-free property can be inferred from the degree and clustering coefficient distributions for the graph (i.e. the histograms of the number of nodes with a specific degree or a specific CC). If the CC distribution is linear constant and the degree distribution follows a linear decrease, the above properties are true and the graph can be considered scale-free.

Hierarchical networks behave similarly to scale-free networks. The probability for an edge to be attached to a particular node still increases with the number of edges the node already has, meaning hubs are also favored structures in hierarchical networks. But in contrast to a scale-free network, where the clustering coefficient is constant throughout the network and independent from the degree of a node, in hierarchical networks, the clustering coefficient of nodes with a higher degree is expected to be smaller than of nodes with a lower degree (Barabási and Oltvai, 2004). This can be seen in Figure 2.2C.

An overview of these three network models and their degree and clustering coefficient distributions is given in Figure 2.2.

An interesting phenomenon in this context is the so-called *small-world effect*. By this effect, the network is structure in a way that the overall average diameter (the longest shortest path of all paths in the network) is small, usually for organic networks in the range of three to eight. Any two nodes of the network can be reached from each other through only a small number of steps. The reason behind is that small-world networks have many hubs and these hubs tend to be highly connected to each other on average (Watts and Strogatz, 1998). This means that it is very likely to have a hub in the direct vicinity of a node and since these hubs are connected to other hubs as well, the paths between any two nodes go directly through these hubs. Consider, for example the network of airports and plane routes. Every city in the world can be reached by any other city in the world by a maximum number of transfers to other planes of about 3-4. The reason is that large international airports that fly to many destinations are the hubs in those networks and result in this small-world property. Many real-world

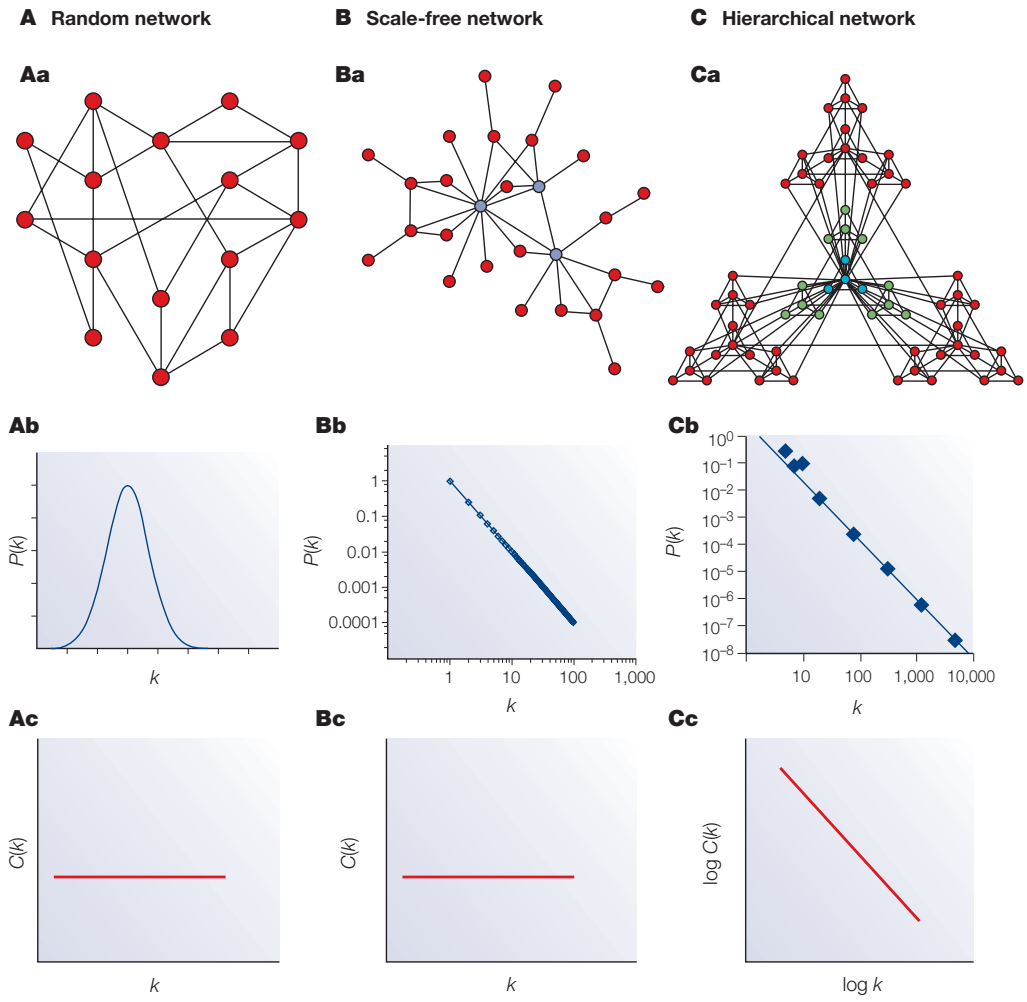


Figure 2.2: Network models. Three models are common: Random networks (Aa) where the degree distribution (Ab) shows a random distribution and the clustering coefficient distribution is linear (Ac). Scale-free networks (Ba) where the degree distribution shows an exponential decrease for higher degrees (Bb) and a linear clustering coefficient (Bc). Hierarchical (scale-free) networks (Ca), where both, degree and clustering coefficient distributions show an exponential decrease (Cb, Cc). Adopted from Barabási and Oltvai (2004).

networks have this small-world property, such as friendship networks[3] or biological gene-regulatory networks. Interestingly, the small-world effect was first described through an experiment by Travers and Milgram (1969), where the authors conducted a survey for the connectedness of people of the world and posed that every human knows every other human in this world through around six other people. Recent investigations, however, seem to decrease that number. Facebook conducted a study in 2008 and in 2011 and it seems that the average diameter of the Facebook graph decreased from 5.28 to 4.74 (Barnett, 2011), which can mean two things: either the number of hubs increases or the degree of hub nodes increases. The last phenomenon could be explained by a desire of some people that stockpile friendships with other people that they usually do not know well.

2.1.2 Clustering and Community Detection

Clusters, or *communities*, in networks are sets of nodes that are very similar to each other in respect to particular characteristics. Nodes can be clustered according to some property (*conceptual clustering*, Balasundaram and Butenko (2008)) or clustered according to their position within a network and the interconnectedness to their surrounding nodes (see also the clustering coefficient). In this case, the defining characteristic for the cluster would be the distance to other nodes, which is called *distance-based clustering* (Balasundaram and Butenko, 2008). In the following we will use the latter definition for network clusters throughout this thesis unless otherwise noted.

While a complete review of community detection methods would go beyond the scope of this thesis, and has already been published for example in Fortunato (2010), I will concentrate here on some specific algorithms relevant to the presented work.

In biology, we have to deal with the aforementioned hierarchical and scale-free networks most of the time. Their inherent structure shows clearly which parts in the network are more closely connected to each other than to the rest of the network. These tightly connected sub-graphs are called clusters or communities and provide interesting insights into the workings of the complex system that the network represents. As an example, if we had a protein-protein interaction network with clearly visible clusters, this would then indicate a common function for all the nodes within the same community (Zhang et al., 2006). In a network consisting of diseases that are connected through shared affected genes, clusters could identify diseases that have a similar etiology (Goh et al., 2007).

However, as communities are rarely closed communities with no overlaps, methods are needed that take into account that a specific node can belong to more than one group. This becomes evident when considering social networks, where for example nodes are people and edges are interactions between people. Obviously, people are contained in more than one community: they have close friends with whom they interact frequently and they have co-workers with whom they also interact frequently. However, the two communities do not necessarily interact with each other, therefore the original node belongs to more than one community. Or, as a more biological example, proteins can belong to multiple functional communities and execute different tasks throughout the body of an organism depending on their interaction partners. For example, the LRRK2 protein is involved in protein phosphorylation

[3] This lead to some curious "games" such as the "Six degrees of Kevin Bacon" (http://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon), where a player tries to connect any actor to Kevin Bacon in as few links as possible through mutual acting roles in the same movie.

and regulation of GTP catabolism. Because it is a multi-domain protein, different domains can execute different tasks within the cell, the overall protein can be involved in more than one function (Smith et al., 2006).

The Clique Percolation Method

In the last couple of years, detecting overlapping communities has become quite popular. An introduction to some of the methods can be found in Fortunato (2010, Section XI). One method, however, has become quite important in biomedicine and lead to the development of the CFinder software package[4]: The *clique percolation* method.

This methods detects communities with the help of k-cliques, cliques with k nodes that are fully connected to each other. They define that cliques are adjacent to each other if they share $k-1$ nodes. A community is then, by their definition, a union of k-cliques that can be reached from each other through adjacent k-cliques (Derényi et al., 2005; Palla et al., 2005). For smaller values of k, for example three or four, the method visualizes communities in graphs that are tightly interconnected with each other.

The CFinder package will be used in Section 6.4.1 for evaluation.

For obvious reasons an in-depth introduction to community detection methods cannot be given in the scope of this thesis. For interested readers, the article by Fortunato (2010) gives a good overview of different clustering and community detection and can be used as a reference for various such algorithms.

2.2 Cloud Computing

> "[...] computing may someday be organized as a public utility just as the telephone system is a public utility [...] The computer utility could become the basis of a new and important industry."
>
> *John McCarthy, 1961*

As early as in 1961, John McCarthy[5], the inventor of the Lisp programming language[6], dreamed of a world where "computing may [...] be organized as a public utility". The concept of having computer resources, such as computation, storage, or other services as devices that can be rented, proved to be a very early glimpse into the future we are now in and laid the basis for the concept we now know as cloud computing (Armbrust et al., 2009; Farber, 2002; Lifschitz, 2011).

In the following I aim to provide a short definition of "Cloud Computing" because of the implications this has on the following chapters. The base technology that is used for the development of the tools

[4] http://cfinder.org/
[5] http://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist), accessed 2012-04
[6] Lisp is one of the oldest functional, high-level programming languages still in use, invented in 1958 by John McCarthy. For more information, see http://en.wikipedia.org/wiki/Lisp_(programming_language)

throughout this thesis has its roots in the concept of cloud computing. This, combined with the earlier mentioned fact that data increases tremendously, I introduce it here in oder to lay the basics for the later technological advancements.

Generally speaking, *Cloud Computing* can be best defined as dynamically scalable, often virtualized services over the Internet. When experts were asked, how they defined cloud computing, each of them gave a different answer, ranging from highly emotional ones to more technological ones (Geelan, 2009). At some point, the National Institute of Standards and Technology (NIST) published a standardized version of a definition for cloud computing (Mell and Grance, 2009) that has been universally accepted later. The NIST's definition states:

> "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." (Mell and Grance, 2009)

Their definition classifies Cloud Computing into three *service models* that can be offered by cloud providers (companies that offer those services).

Consumers (users) often access software that runs on a cloud computing infrastructure by some company, which is called *Software as a Service (SaaS)*. In other words, the user of the software does not care where the software is installed or what operating systems are used. For the user, this is a completely transparent process as well as roll-out of new features and updates. Examples for SaaS include popular internet services such as YouTube[7] or Google's GMail[8] service.

One step below that is the platform that can be offered through *Platform as a Service (PaaS)*. When accessing platform modules, the cloud provider supplies users with a specific set of tools to help develop and run applications on a specific infrastructure. Examples for such platforms include Microsoft's Windows Azure[9] platform or the Apache Hadoop ecosystem (see Section 2.3.3).

On the lowest level, cloud providers can offer *Infrastructure as a Service (IaaS)*. In this model, the user has full control over a set of virtualized computers on top of a cloud infrastructure. While the user cannot control the hardware of a machine directly, he can chose which operating system to install and everything up from that. Examples include Amazon's Elastic Compute Cloud services EC2[10].

The three cloud computing service models SaaS, PaaS, and IaaS can be deployed in several different scenarios, depending on the needs of the user.

Private clouds belong to a single organization and provide fine-grained access control to every item of the hard- and software stack. Private clouds are basically servers that can either be hosted on-premise or off-premise. They belong to the company that uses them and are used exclusively.

By contrast, *public clouds* are basically a pool of cloud services that are sold to the public and can be used by anyone. An example for this is Amazon's Web Services (AWS) that include their EC2 system, which is an IaaS service offered through a public cloud.

[7] http://www.youtube.com, accessed 2012
[8] http://mail.google.com, accessed 2012
[9] http://www.windowsazure.com/, accessed 2012
[10] http://aws.amazon.com/, accessed 2012

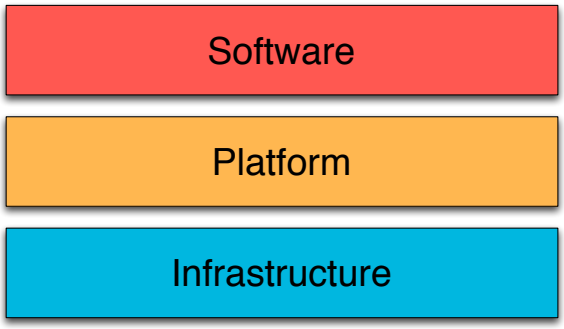


Figure 2.3: Cloud computing service models. The most basic infrastructure of a computer/data center of an organization is hardware. Usually, a virtualization layer provides this infrastructure as a service component. On top of the infrastructure runs the platform. The platform consists of the operating system (not accessible by the consumer) and standardized libraries. Finally, software as a service abstracts all other parts and offers the consumer the use of specific applications.

Any combination of private and public clouds is called a *hybrid cloud*. It combines the advantages of a cheap and scalable public cloud with the security measures of a private cloud and compartmentalizes them into a single system. The compartments can interact with each other through standardized technology. Examples include Amazon EC2 on the public side and an on-premise system outfitted with the Eucalyptus Cloud System[11].

Finally, the NIST defines five essential characteristics for Cloud Computing.

On-demand self-service: A consumer of cloud services can acquire resources, such as processing time, processing capabilities, storage, etc on an as-needed basis by interacting with a web browser or similar tool.

Broad network access: Everything is accessible through a network, in public clouds through the Internet.

Resource pooling: Resources in the data center are pooled, meaning the infrastructure supports a multi-tenant model: everything serves multiple consumers. Resources that can be shared include storage, processing, memory, network bandwidth, and virtual machines.

Rapid elasticity: A consumer is able to purchase resources and capabilities of rented systems in any quantity at any time possible. This allows then for a rapid and elastic provisioning.

Measured service: The resources used by a consumer can be monitored, controlled and reported for increased transparency to the provider and the consumer himself.

Some argue that cloud computing is nothing more than repackaged grid computing (Foster et al., 2008). While it is true that there are many overlaps in technology and applications between cloud and grid computing, the difference lies in the essential characteristics mentioned above. Especially the point of rapid elasticity could only be argued successfully, if the underlying technology is built on virtualization. Without virtualization, a user would have no capabilities in transparently ordering processing capabilities in (nearly) every quantity. An overview with other common distributed systems is given in Figure 2.4.

[11] http://www.eucalyptus.com, accessed 2012

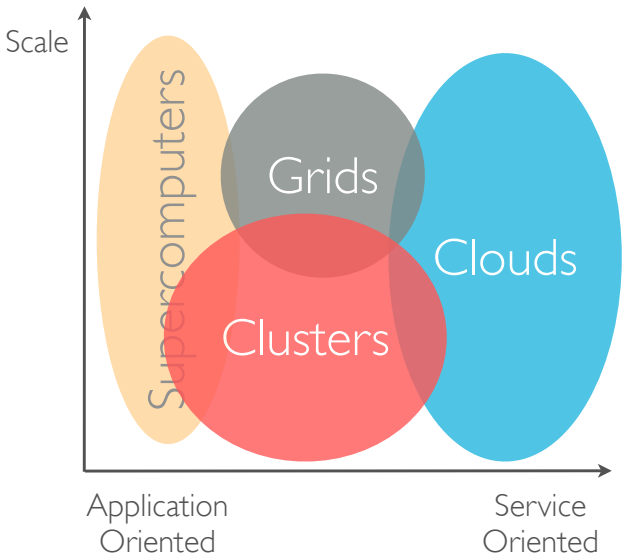


Figure 2.4: Grids and clouds. Clouds heavily rely on the service orientation, offering a consumer as many possibilities and services as possible. A supercomputer on the other hand is a highly specialized piece of hardware not suited for everything. Grids and Clusters are somehow in between. "Scale" does not mean scalability but the amounts at which the technology can be purchased without major problems. Adapted and modified from Foster et al. (2008).

As can be seen later in this thesis, the employment of cloud computing technologies has some particular advantages. A software framework that is build upon a PaaS model can automatically benefit from built-in scalability measures, such as the Hadoop software framework (Section 2.3.3). The concept of acquiring computational power on-demand and in necessary amounts is a big factor when considering data- and processing-intensive applications not only in the area of text mining but in all areas of biology.

2.3 The MapReduce Paradigm

> "So what are you waiting for? Go find some reasonable gigantic data processing problem and mapreduce it into insignificance."
>
> *Shrutarshi Basu, May 2008, http://bytebaker.com*

When Google was formally founded in 1998, their first search engine prototype was hosted in a garage and built from some cast-off hardware, which was enough at that time. Without going into too much detail on the history of Google or the Internet, both became larger and larger over the next years, which created a need for more sophisticated and powerful hardware in order to store the index of the Web. Soon they discovered that this can only be achieved by integrating many machines into clusters and these clusters into data centers. However, as hardware is not impeccable, a solution had to be devised how to cope with machines that constantly have errors and fail. For its search engine, Google was on the look for a technology that works on a cluster architecture and copes well with hardware failures (Shankland, 2008a,b). A study of the failure rates in big data environments by Google (Pinheiro

et al., 2007) showed that over 90% of all data is stored on magnetic disk arrays (hard drives). The authors report that hard drives are the single most common type of failing hardware. Additionally, there seemed to be no consistent indicative pattern to predict the failure of a hard disk, which makes it obvious that the overlying platform and software that uses these machines has to deal with failing drives at runtime

A further major point was that Google needed to be able to efficiently process the Web and needed to have specialized resources for handling large data. If any computation (a task) fails because of a hardware error, there should not emerge any side effects and the computation should be seamlessly restarted in order to complete processing.

The same considerations are basically valid for data processing in biomedicine. Either for text mining on publication data or any other data such as DNA sequence reads, a powerful hardware and software architecture is vital to solving any problems and being able to compute anything.

In their paper "MapReduce: Simplified Data Processing on Large Clusters" by Dean and Ghemawat (2004), Google present their take on the problem in form of the cluster framework MapReduce. The idea to it came from two constructs that can be found in functional programming languages, particularly in Lisp: *map* and *reduce*. A map-function is a higher-order function that applies a given primitive function to a list of values and returns another list as a result. A reduce-function is also an higher-order function that works by recursively processing the elements of a recursive data structure (such as a list) and then returns a single value. In the MapReduce framework the map and reduce functions are similar but not identical to their functional programming counterparts. While a map function basically computes what its functional programming counterpart does, the reduce function in MapReduce does not necessarily return only a single value but can also return a list with less or more values depending on the use-case.

What is done, in other words, is a logical operation (map) on each input record in order to produce some intermediate key-value pairs and then a reduce operation that takes all intermediate values with the same key in order to produce some appropriate end result. The basic idea behind the framework is now to split up large data sets and compute single chunks on different servers via the map methods and combine intermediate results to a common result in the reduce phase.

Figure 2.5 graphically shows how the MapReduce framework works. A program consists of one or more jobs that each have one map function and an optional reduce function. Input data is split up into approximately even-sized chunks of data that each run a separate instance of the map function. Data is therefore processed in parallel depending on the number of map tasks. The map function outputs the data which is then sorted by the MapReduce framework before entering the reduce phase. Here, another operation attempts to combine the input into a central output.

Algorithm 2.1 shows an example MapReduce program. This program (in pseudocode) is considered the MapReduce Hello-World[12] example and counts the frequency of words in the input data. A list of documents is first split into different sets. For each document in a set, the map phase tokenizes the input document text into separate words and outputs a count of one for each obtained word. The reduce phase in turn gets the word and the list of ones that were obtained by any map for this word. It sums these ones up and emits the total count for every particular word. Between the map and the

[12] A Hello-World program is most times the simplest possible program in any particular programming language. It is often used as one of the first examples when learning a new language. Here it is used to introduce the simplest program with the MapReduce paradigm. See Wikipedia (http://en.wikipedia.org/wiki/Hello_world_program) for more examples on Hello-World.

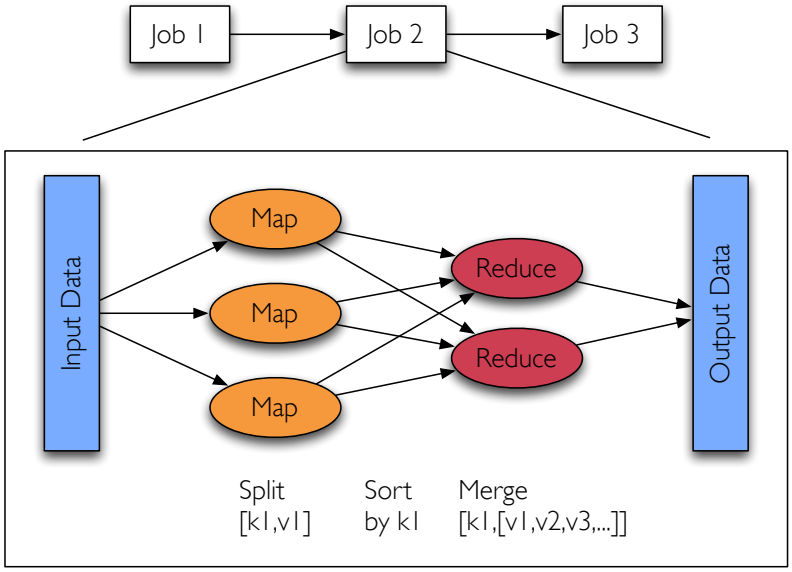


Figure 2.5: Overview of the MapReduce framework. Every job has input and output data. The input data, while usually huge, is split up into multiple chunks of data of approximately the same size. A predefined map function is then executed on every chunk in parallel due to a distribution onto multiple computers (a cluster). Results of this map-phase are sorted and combined to a single output data result in the reduce phase of a job.

function MAP(key k, value d)
1: **for each** word w in d **do**
2: EMIT(w, 1) count all words and emit together with a 1
3: **end for**

function REDUCE(key k, list[value] c):
4: $r \leftarrow 0$
5: **for each** counts v in c **do**
6: $r \leftarrow r + v$ sum up all counted 1
7: **end for**
8: EMIT(k, r)

Algorithm 2.1: The MapReduce Hello World example. The *map*-phase has the name and the contents of a each document as its input key-value pair. It then emits a single one for each word it encounters within this document. In the *reduce*-phase, the word and a list of all the ones of all maps is the input. The ones are added to each other to produce a final number of occurrences of each word.

reduce phase, all data is sorted in a way that the reducer gets all the frequencies that belong to a particular word. This makes sure that there will be exactly one output number for each word.

This algorithm can work the way it is presented due to the intermediate sorting step between the map and the reduce phases. This sorting guarantees that the reducer gets all values for the same key in order to process them further within the same reduce method call.

2.3.1 MapReduce Algorithm Design

One basic design principle behind parallel algorithms is the so-called *divide-and-conquer* approach. The idea is to partition a larger problem into smaller sub-problems. If these sub-problems are logically independent from each other, they can be addressed in parallel by different workers, such as processor threads, processor cores, processors or full machines in a cluster (Lin and Dyer, 2010). Afterwards, results can be merged again into a single output. Obviously, such an approach is only necessary and useful, if the problem at hand is large in terms of data volume or computation.

The MapReduce framework offers application developers an environment where they can implement large-scale applications independent of any hardware and node-to-node communication issues. The functional programming paradigm together with the fact that MapReduce's input and output is always in key-value format, leads to the fact that not everything can be processed with it. Some algorithms can be directly transformed, others have to be rewritten and rethought, and others simply cannot be forced onto the MapReduce framework (yet).

Some key properties need to be considered when developing MapReduce algorithms. The key idea always is to scale out, instead of scaling up. In other words, if for some problem the computing power at hand is insufficient, more machines should be thrown at the problem instead of bigger, more powerful machines[13]. The reason is that for any particular MapReduce problem, the parallelization step is more important than the execution time of a single item, because of a limited amount of processing power for a single machine when considering the acquisition price. Put differently, at some point the increase in processing power and hardening against failures for a single machine will increase faster than a scale-out solution where individual machines have lower power. As an example, at the time of writing this thesis, an Intel Xeon processor with four cores at 1.8 GHz cost about 200 €. The same CPU with eight cores cost about 1200 €[14]. In other words, double the power for a CPU would increase the cost six-fold, while increasing the number of machines would also only double the cost. In this consideration I obviously did not include the cost for other components of a server, but they are negligible compared to the six-fold cost increase of the CPU.

Another point is input and output of data (I/O) to and from hard drives and especially through the network interconnecting different machines. This is definitely a limiting factor because of which the MapReduce framework tries to execute any algorithms only on the data that is local on the current machine.

[13] Obviously, code optimizations and better algorithms should be considered first. However, if the input data is sufficiently large, better algorithms will not work and the hardware has to be upgrade in one way or another.
[14] http://www.transtec.de, accessed 2012-09-27

Despite that, the advantages are obvious: the MapReduce paradigm provides seamless scalability. When more machines are needed, they can be plugged in and the application should scale automatically because the underlying framework takes care of that. Also for that reasons, everything that is done in a MapReduce job should obviously limit calls to external resources such as databases or anything that is not locally available.

The trend to parallelize and distribute computations became famous only in the last couple of years for commodity machines and regular desktops. The reason behind is probably an increased interest for multi-core architectures by processor manufacturers such as Intel and AMD.

In MapReduce, a developer is confronted with mapper and reducer methods (or classes) that have the following signatures (squared brackets denoting a list, round brackets denoting a tuple):

$$\text{map: } (k_1, v_2) \rightarrow \left[(k_2, v_2)\right]$$
$$\text{reduce: } (k_2, [v_2]) \rightarrow \left[(k_3, v_3)\right]$$

Basically, there are two method signatures: map and reduce. The map methods have a single key k and a single value v as input and also produce a single key and value as output. The reduce methods get a single key with a list of all values that have the same key as input and again output a single key and a single value.

For developers, difficulties can arise when attempting to transform common algorithmic problems to MapReduce. As all such methods have to obey the stated method signatures, not always an efficient implementation of other algorithms can be found immediately. These transformations have been subject to multiple studies. Zhao et al. (2009), for example, implemented a parallel k-means clustering, Verma et al. (2009) out-scaled genetic algorithms to work on MapReduce, and Matthews and Williams (2010) and Sarje and Aluru (2010) investigated how to transfer algorithms working on tree structures to MapReduce.

In the following, I will describe some popular *design patterns* for MapReduce applications that have proven to be useful in many applications. Most of the descriptions are adapted from the book by Lin and Dyer (2010, Chapter 3) unless otherwise noted. The principles described here have been implemented in various jobs for the text mining system Excerbt, but the names for these design patterns have only been taken recently out of the book.

In-Mapper Combining

A common problem when emitting key-value pairs from the map phase to the reduce phase is intermediate result synchronization. For the reduce phase all values with the same key need to be on the same server. In order to work, data has to be (a) written to disk and (b) transferred over the network . The obvious bottleneck here is I/O. The idea therefore is to locally pre-aggregate intermediate results as far as possible. A smaller number of results will be produced from a larger list when those results are still in memory. This ultimately creates much more efficient algorithms.

Consider the word-count example from Algorithm 2.1. For each word in a sentence, the mapper emits that word as the key and the single natural number one as the value. The reducer then sums up all numbers and returns the final absolute frequency count of a particular word. The approach is

obviously flawed since for any document that is subject to the map phase, some words might occur multiple times and would be emitted separately. This increases the number of key-value pairs the reducer has to take care of tremendously for a very large data set. But, the algorithm can be enhanced by specifying an additional so-called *combiner* that basically is some kind of mini-reducer with the same method signature as a reducer. The combiner runs directly after the map phase which can have significant performance advantages. Due to the architecture of the MapReduce framework, data between map and reduce will be sorted according to the map output key, such that all values with the same key will be processed within the same reduce method call. This makes it necessary to transfer data through the network. A combiner, however, can diminish the amount of work the intermediate sorting and reduce steps have to perform as it runs on the same machine as the mapper.

In the example, a combiner could sum up all ones for equal keys that this mapper has processed before emitting them to the reducer and therefore before the sorting step which distributes the key-value object for the reducers to the correct machines in the cluster. This then results in larger counts for the values of the intermediate result but less of them. In case the question why we cannot simply use the length of the values in the reduce phase instead of adding all the ones came up, the use of a combiner makes the answer obvious.

Instead of combining *after* the map phase, the mapper could be rewritten in order to allow combining *during* the map phase. If in the initialization phase of the mapper, a map data structure is set up, it can hold intermediate results until all map tasks are completed. The mapper would then emit the contents of the map in its shutdown (close) phase and only output unique words for all documents the mapper has processed. This represents an optimization of runtime efficiency. The algorithm for the word-count example is outlined in Algorithm 2.2. If we combine the values after the mapper, we have a lot of disk-saving overhead between the mapper and the combiner since all results of a map phase must be persisted on disk before being processed again.

method INITIALIZE:
1: $H \leftarrow$ new MAP — initialize a map that holds intermediate data

method MAP(docid a, doc d):
2: **for each** term $t \in$ doc d **do**
3: $H\{t\} \leftarrow H\{t\} + 1$ — increase value in map instead of emitting
4: **end for**

method CLOSE:
5: **for each** term $t \in H$ **do**
6: EMIT(term t, count $H\{t\}$) — emit values in map that have been pre-combined
7: **end for**

Algorithm 2.2: In-mapper combiner pattern for the word-count example. Given are the three methods of the mapper class: initialize, map, and close. In the initialization phase, a map structure is initialized. The map method then outputs into this map phase instead of emitting key-value pairs. The close method then has already an aggregated version of all the key-value pairs of this run and simply can emit the contents of the map. Adapted from Lin and Dyer (2010).

This version is more efficient than a simple combiner. Network traffic is limited in both implementations, but in the in-mapper combiner pattern the number of intermediate results to be emitted is greatly reduced, therefore limiting the amount of data, the MapReduce framework has to take care of.

This design pattern, however, has one disadvantage. The memory that is assigned to the mapper function is limited. If the map data structure that holds intermediate keys and their counts is too large, because too many documents were processed in this class, the data structure might not fit into memory anymore, resulting in a failed job.

The use of the pattern therefore highly depends on properties of the data being processed. If many local values have the same key, it might be useful to use it. If not, the combiner and in-mapper combiner patterns might not be useful and should be omitted.

Pairs and Stripes

Another, often occurring problem becomes evident when dealing with matrices in MapReduce. If a matrix has m columns and n rows, it has $m \times n$ cells and an $O(nm)$ space requirement. If $n = m$ the space requirement is quadratic for the number of rows or cells, $O(n^2)$. Depending on the sparseness property of the matrix, a more efficient representation can be reached, by not including empty cells into the representation. In the worst case, the matrix is completely dense. Regardless of the actual representation, if such a matrix fully fits into memory, any computation is straight-forward. For sufficiently large matrices where the contents does not fit completely into memory anymore, any algorithm cannot be reasonably time efficient when intermediate data has to be stored on disk instead of in memory. Matrices can grow in either rows, or columns, or both. If they only grow in one dimension, the space requirement for computations increases linearly, for both dimensions it grows quadratic.

This is where the *pairs* and *stripes* design patterns come as useful approaches for MapReduce algorithms that are based on matrix operations (see Figure 2.6), specifically dense matrices. The MapReduce system grows linearly in computing and storage power with a linear added number of compute machines.

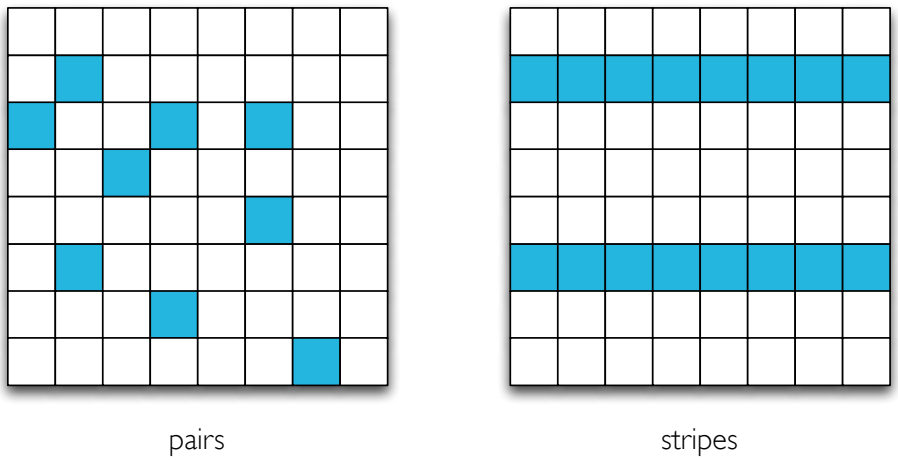


Figure 2.6: Pairs and stripes MapReduce design patterns. The pairs design pattern computes matrices by emitting for each cell a key-value object, the stripes design pattern emits a key-value object for each row (or column).

The next few paragraphs will explain these design patterns on the basis of computing co-occurrence matrices. A co-occurrence is defined as the combined occurrence of two components within a fixed

region, for example a tuple of any two words within a sentence. In order to compute the frequencies for every word tuple within a text corpus, the following methods can be employed.

The pairs design pattern basically computes the matrix by specifying a mapper that emits for each chunk of text within one map method call the two co-occurring words as the key (e.g. design and pattern) and the count (a single one) as the value. The reducer then sums up all those counts and in turn emits a value for each cell in the matrix. It is obvious that this algorithm generates a massive amount of key-value pairs. In the average case, where each sentence in the corpus has n words, with m sentences, in total $mn(n-1)$ key-value pairs have to be emitted. Put differently, if the matrix has u rows and v columns, and the final value in a cell is denoted by $c(u,v)$, the algorithm emits $\sum_u \sum_v c(u,v)$ values.

The stripes version of this algorithm on the other hand works not by emitting single cells, but entire rows. The mapper returns a single word as a key (e.g. design) and all co-occurring words within the document as a map as the value. The reducer then simply has to add all the different maps for the same words and produce some final map. The study by Lin (2008) measured the runtimes of the pairs and stripes algorithms for various test sets and various matrix sizes and confirmed by linear regression that this improvement makes the stripes version much faster in every case. One reason is that there are a lot less key-value pairs to be sorted by the framework than in the pairs version.

Both algorithms can profit from the previous combiner and in-mapper combiner design patterns. Depending on the application the pairs approach might be more useful or the stripes version. In case both versions can be used, the stripes version is always faster.

Other MapReduce Design Patterns

The list of existing MapReduce algorithm design patterns obviously does not end with the in-mapper combiner and pairs-and-stripes techniques. Just to name a few others: *Order inversion* is an approach where the computation of a specific problem is turned into a sorting problem so the MapReduce framework can take care of it without large overhead. This is a very general approach and is used extensively in any MapReduce application. *Value-to-key conversion* can be used for sorting the values that go into the reducer. If the list of reducer input values is large, a guaranteed sort-order could be desired. Finally, Lin and Dyer (2010) also give an introduction how to implement various forms of relational joins with MapReduce. These algorithms are implemented in ad-hoc query languages such as Hive (see Section 2.3.4). More design patterns and a more detailed explanation of them can be found in the book Lin and Dyer (2010).

2.3.2 MapReduce Graph Algorithms

As explained, the design and implementation of MapReduce algorithms can be a daunting task. The prerequisite that input and output of map and reduce methods must be in key-value pair format makes this especially tedious for algorithms handling graphs or networks. Traditional graph algorithms, such as Dijkstra's algorithm (Dijkstra, 1959) or depth- and breadth-first-search (Even, 2011; Joyner et al.,

2012), cannot be used in MapReduce without significant adaptation due to the nature of distributed, parallel computing.

In mathematics, graphs are usually represented as *adjacency matrices*. These matrices contain the nodes as row and column identifiers and the cells contain either a 1 if there is an edge between the two nodes or a 0 if there is no edge. For undirected graphs, the matrices are usually symmetrical along the diagonal. While this representation is useful for linear algebra because of easy manipulation of the graph through row or column operations, it is not suited very well for the use in computer programs because of an obvious $O(n^2)$ space requirement. When the number of a graph's nodes or edges goes into the millions, by then the problem becomes relevant at the latest.

However, there is a property of graphs that can be exploited for our purposes, the *sparseness*. We call a graph sparse, if the number of actual edges is *far* smaller than the number of possible edges (Black, 2004). In other words, most of the values within the adjacency matrix of a sparse graph are zero, i.e. the nodes are not linked through an edge. Graphs that are not sparse, are called *dense*. Most naturally occurring networks (scale-free networks) are sparse. Therefore, another graph representation, the adjacency list, is a viable alternative to the matrix representation. This representation gives the direct neighbors of any node in a list representation and therefore omits the massive overhead of representing all possible edges regardless of whether they exist or not. In contrast to adjacency matrices, computations on lists are more difficult, since the algorithm only has a local graph structure around the center node available for processing and cannot use global matrix operations (see Figure 2.7).

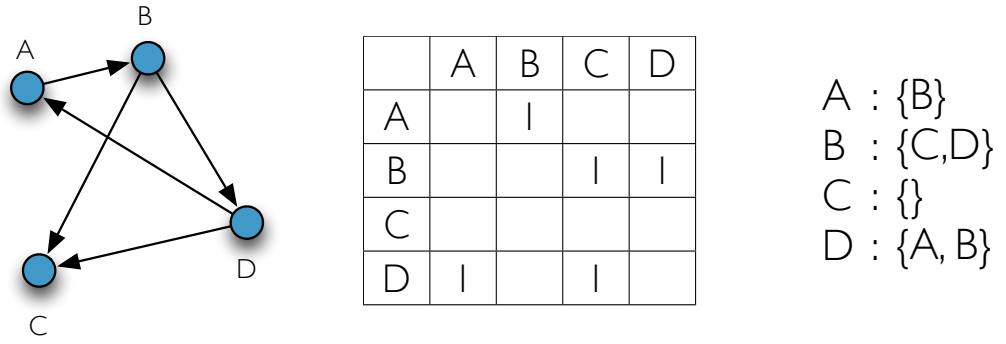


Figure 2.7: Graph representations. A sample graph with 4 nodes and its representation in matrix form (middle) and adjacency list form (right). Only outgoing edges are displayed in the list, incoming edges, however could also be stored.

Consider, for example, the problem of finding the shortest path of a single node to all other nodes (*single source shortest path*). The traditional approach to this problem would be to use Dijkstra's algorithm that attempts to solve the problem through a graph traversal of the edges with the shortest distance. Without going into further detail of the algorithm, the problem is that it operates sequentially on each node. The key to the algorithm is to store a globally sorted list of nodes by current distance to the source node (Lin and Dyer, 2010). In MapReduce, on the other hand, this is not possible as due to the architecture we can[15] only operate on local data (as previously learned). What we can do with MapReduce is to implement an efficient brute-force[16] method in order to completely search the graph

[15] or should if we want to be efficient
[16] http://en.wikipedia.org/wiki/Brute-force_attack, accessed 2012-06-01

for paths in a parallel fashion. The idea is to use a parallel version of the breadth-first search algorithm in order to start from the given source node and to iteratively increment the "search frontier" (Lin and Dyer, 2010) by one step in each iteration, that is to basically follow all the edges from a given starting point one step at a time until all nodes of the graph have been visited.

In Algorithm 2.3 a version of a parallel breadth-first search is outlined. The basic approach of breadth-first search is to start from a single node and cycling through the neighbors and neighbor's neighbors etc until every node in the graph is visited once. In MapReduce, one call of the map and the reduce method for a graph basically moves the frontier covering nodes one step further. This implies that it is an iterative process[17] and that the number of iterations is equal to the integral diameter of the network[18]. In most networks in biology this can still be performed efficiently, because the small-world effect (Section 2.1.1) which uses the diameter definition, limits the number of iterations usually to a very few.

function MAP(nid n, node N):
1: $d \leftarrow N$.DISTANCE
2: EMIT(n, N) — pass along graph structure
3: **for each** nodeid $m \in N$.ADJACENCYLIST **do**
4: EMIT($m, d+1$) — emit distances to reachable nodes
5: **end for**

function REDUCE(nid m, $[d_1, d_2, \ldots]$)
1: $d_{\min} \leftarrow \infty$
2: $M \leftarrow \emptyset$
3: **for each** $d \in$ counts $[d_1, d_2, \ldots]$ **do**
4: **if** ISNODE(d) **then**
5: $M \leftarrow d$ — recover graph structure
6: **else if** $d < d_{\min}$ **then**
7: $d_{\min} \leftarrow d$ — look for shorter distance
8: **end if**
9: **end for**
10: M.DISTANCE $\leftarrow d_{\min}$ — update shortest distance
11: EMIT(m, M)

Algorithm 2.3: An algorithm for parallel breadth-first search. The map phase basically emits the distance to each reachable node from the key node for a distance of 1. The reducer then uses these distances to compute the shortest distances to each of the nodes as the reducer key is the target node with a list of distances. Adapted from (Lin and Dyer, 2010, Chapter 5).

In order to work, intermediate results from the map to the reduce phase have to have the graph structure of the node attached. Through MapReduce's shuffle and sort, we make sure that the algorithm works.

Various algorithms have been adapted to work with Hadoop's MapReduce framework. Cohen (2009), for example, introduced algorithms for the computation of clustering coefficients via counting triangles

[17] the processing of individual operations has to be configured elsewhere, i.e. in the code executing the mapper and reducers
[18] which basically also gives a method to determine the diameter of the network

and rectangles which lead to basic clustering methods for graphs. This and similar approaches are explained in more detail in Section 6.

One problem with MapReduce in graph processing, though, is also it's greatest strength: to minimize network transfer. A partitioning of a graph into completely independent sets of partial data is impossible because of the linked structure of graphs. However, some workarounds have been proposed by multiple studies. Cohen (2009) stated that he found efficient solutions in MapReduce for most graph-algorithmic problems that rely on message passing between individual nodes, but that any algorithm relying on depth-first search probably will have poor MapReduce performance. Another point is that reduce functions cannot be executed directly after the other. In a bigger job list, a map function always has to precede the reduce function, creating a computational overhead that could theoretically be avoided. An example of this will be seen in Algorithm 6.4.

Procedures to enhance Hadoop's capability to handle graphs have been suggested (and tested) in multiple studies. Abadi (2011) and others (Bajda-Pawlikowski et al., 2011; Huang et al., 2011; Rohloff and Schantz, 2011) basically describe that the main problem for efficient MapReduce implementations of graph problems is the partitioning of input data. In MapReduce data is usually partitioned by determining splitting input data after specific amounts of contiguous values. A graph that is stored as an adjacency list with lexicographically sorted node labels (i.e. nodes starting with A are before nodes starting with B), would, for example, put all nodes starting with A into the same partition. If this partitioning would be decided by other approaches, such as a clustering-based method, nodes that are tightly interconnected could be put into the same partition instead of different ones. MapReduce algorithms could profit from that because of a limited amount of data that needs to be transferred through the network. Admittedly, work in this area is not yet ready for actual implementation.

Google also have recognized the problem with graph processing in MapReduce. Because of this, they published the ideas to their implementation on graph processing on top of MapReduce clusters, which is called *Pregel* (Malewicz et al., 2010). The ideas presented in the paper have been transferred to an open source implementation by *Apache Giraph*[19]. Pregel and Giraph are basically libraries that abstract the concept of nodes and edges onto the MapReduce framework. Another library that works in a similar direction is the Pegasus project[20]. These three projects, however, were only made available in useable form in the last couple of months. Therefore they are only introduced here to give an outlook into future developments of graph processing in MapReduce.

Other approaches for large-scale graph processing apart from implementations in MapReduce have been published by, e.g. Twitter[21] in their open-sourced library Cassovary[22] (Gupta, 2012). Low et al. (2012), for example, invented their product GraphLab[23] which introduces a library to work with machine learning problems on graphs.

More examples for graph algorithms are given in Sections 6.2 and 6.4.2, where I will explore the possibilities to compute the clustering coefficient distribution of a large network and present an algorithm for large-scale local clustering.

[19] http://incubator.apache.org/giraph, accessed 2012-06-01
[20] http://www.cs.cmu.edu/~pegasus, accessed 2012-08
[21] http://www.twitter.com, accessed ,
[22] https://github.com/twitter/cassovary, accessed 2012-06-01
[23] http://graphlab.org/, accessed 2012-06-01

2.3.3 Hadoop

Apache Hadoop[24] is an open source implementation of the MapReduce framework, written in Java. It was originally developed as an engine to run Nutch[25], a project that attempt to index the web White (2009) similar to the Google search engine. In its initial days Nutch struggled as a working search-engine but soon became a successful project after Google published its two articles Ghemawat et al. (2003) and Dean and Ghemawat (2004) and Nutch implemented the concepts introduced in those papers. Soon, Yahoo became interested in the technology, and the part concerning distributed processing of data was split off of Nutch into its own project and called Hadoop. Regarding a classification in the cloud computing services models, Hadoop is a representative of the platform of a service model.

Hadoop consists of 3 main components: Hadoop Core, which provides resources used by all Hadoop components, Hadoop MapReduce, an implementation of the MapReduce framework to work on big data, and Hadoop DFS (HDFS) a distributed filesystem necessary in order to provide an adequate data store for the huge amounts of data processable by MapReduce on a cluster architecture.

On an architectural level, Hadoop MapReduce is implemented as a subsystem of server processes. A master server, called the *JobTracker* is responsible for resource management and job submission and distribution. Slave nodes run a service, called a *TaskTracker* that is responsible for actually running the task. Each TaskTracker can run a preconfigured amount of map- or reduce-jobs simultaneously (individual tasks). Each task is started in its own JVM[26] and is therefore independent from other tasks running on the same server. For each input data segment of a single task, the TaskTracker calls the `map()`-method of the implementing Java code exactly once. After all mappers are finished, the output data is sorted lexicographically by the output data key, such that values with the same key will be sorted to the same machine and executed within the same reduce task. After all output data is sorted and copied to the respective machine, the reduce system starts in a similar way as the map system.

HDFS - The Hadoop FileSystem

In order to process data in a distributed fashion where the main strategy is to distribute big data onto multiple machines, a central management system for handling storage and access of the data in a reliable way, called a file system, is necessary. Hadoop contains the file system *HDFS* (Hadoop Distributed File System), which was modeled after Google's distributed filesystem GFS (Ghemawat et al., 2003).

The broad idea behind HDFS is that data is split into roughly equally sized chunks of data which are in turn distributed roughly equally among the machines of a cluster. The chunk size is by definition usually in the range of 64MB to 256MB depending on the use case. This implicitly makes the file system perform well for large files but is inefficient when handling many small files. If a file is smaller than the chunk size (small files), the operating system nevertheless needs to allocate the size of the chunk amount of storage capacity from the underlying operating system. It is clear that this strategy is relatively costly if the number of small files grows too much.

[24] http://hadoop.apache.org
[25] http://nutch.apache.org
[26] The Java Virtual Machine, responsible for running Java programs in Java Bytecode, see Wikipedia (http://en.wikipedia.org/wiki/Java_virtual_machine) for details.

In order to counter the effects of failing hardware (hard disks), the distributed file chunks are replicated at least three times across the cluster. If one chunk went missing without replication, the complete file would be corrupt and unusable most of the times. The replicated chunks are located on different servers (or at least different hard drives) such that the failure of some few components won't affect overall stability of the system.

The Hadoop DFS is handled by a master process, called the *NameNode* that is responsible for assigning which chunk goes to what server and maintaining that information. On the slave side, so-called *DataNodes* are responsible for actually interacting with the underlying operating system and providing low-level file access methods (Figure 2.8).

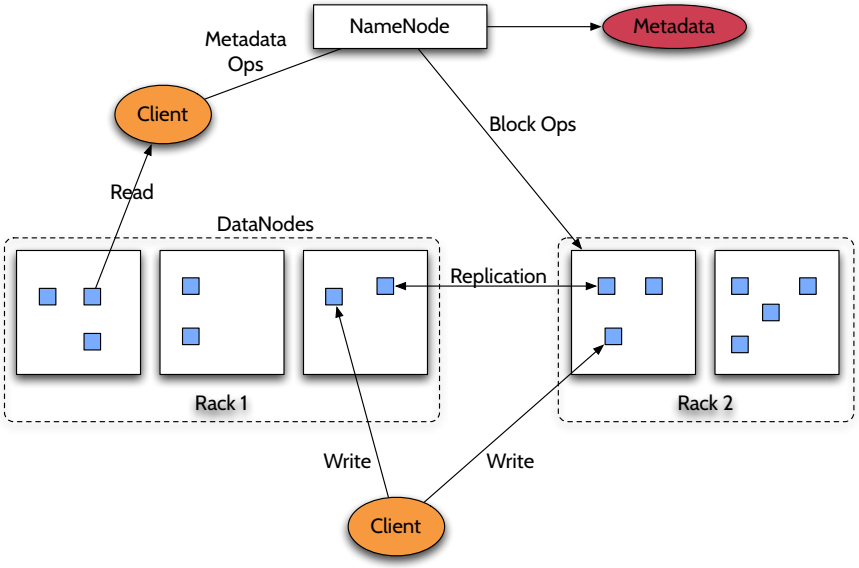


Figure 2.8: The Hadoop file system (HDFS). Clients contact the NameNode for data operations. The NameNode stores references to data chunks on the respective client DataNode servers. Clients then can read and write directly from and to DataNodes after they have communicated with the NameNode. Adapted from (Borthakur, 2010).

All MapReduce processes in Hadoop read and write their data to and from HDFS. The file system provides a layer similar to regular file systems and cares for transparent behavior. Clients to access data in HDFS exist for various programming languages and even a FUSE[27] implementation to mount HDFS into a directory on any Linux client for users exists.

2.3.4 Hive

When implementing algorithms that work on big data, the MapReduce framework offers great possibilities for software developers to optimize everything to the littlest detail. However, working with the data in a higher-level language such as SQL[28] with regular databases, is still a desirable goal. Hive[29] is a data warehousing system built on top of Hadoop MapReduce and HDFS that sets upon that goal. It

[27] http://fuse.sourceforge.net/, accessed 2012-06
[28] Structured Query Language, see Wikipedia (http://en.wikipedia.org/wiki/SQL) for details.
[29] http://hive.apache.org

provides users, which are most often data analysts and not engineers, a declarative language with a syntax close to SQL.

Hive was initially developed by Facebook[30] and is now an open-source top-level project of Apache. Hive is built around the same concept as a relational database and offers the possibility to view data stored in HDFS via tables with a predefined structure. It then offers on top of those tables the same possibilities that plain SQL offers. Select, Insert, Update, and Delete statements can be used to manipulate the data as well as joins and aggregations (group by).

When executing a statement in Hive, such as, for example,

```
SELECT users.gender, count(users.id) FROM users GROUP BY users.gender;
```

the query code is first parsed using a sophisticated system. Different parts of the query are translated into MapReduce jobs that in turn contain optimizations regarding execution (Thusoo et al., 2010). The above query, for example would need one Mapper and one Reducer. First, for each gender, a count of the user ids is emitted in the map phase and subsequently those counts are added within the reduce phase. This makes it obvious that this is only intended for big data, not for small data sets stored in HDFS as the cost of calling and executing MapReduce on small data sets is very high in terms of job setup time.

Basic examples for applications in Hive include summarization/aggregations of recurring data, ad hoc analyses, data mining, spam detection, etc. (Fac, 2008). Hive is used extensively throughout this thesis, namely most of the analyses made for Chapter 5 were computed ad-hoc with Hive queries as an implementation in plain Java MapReduce can be very time consuming.

2.3.5 Summary and Outlook

In this section, I have outlined the Big Data framework MapReduce. At the beginning of my thesis, Hadoop was still a relatively small project with a small but select community. Since then, the project gained a massive momentum and can now be considered as one of the leading technologies for the 21st century as the long list of Hadoop users[31] with some very famous members such as Facebook, Twitter, IBM, Dell, LinkedIn etc. proves. A new demand for Big Data Science has emerged and contributed to the increasing popularity of the framework that could not have been foreseen two to three years ago.

Hadoop is basically a cloud computing platform as a service framework. It offers a toolset that developers can program against. Most of the times, Hadoop is used as a private cloud, but companies such as Amazon also offer EC2 instances preconfigured for Hadoop and ready to use.

Typically, a cluster that runs Hadoop has a lot of different server tasks: NameNode, DataNodes, JobTracker, and TaskTrackers. A usual combination for slave nodes is to run a DataNode and a TaskTracker. The DataNode is responsible for HDFS and the TaskTracker for executing a job on top of this data. Most often, also RegionServers (introduced in Section 2.4.1) are running on the same machine. The Hadoop ecosystem consists of a lot more units than could possibly be described here. An overview of the Hadoop ecosystem is given in Figure 2.9.

[30] http://developers.facebook.com/opensource/
[31] http://wiki.apache.org/hadoop/PoweredBy, accessed 2012-06-28

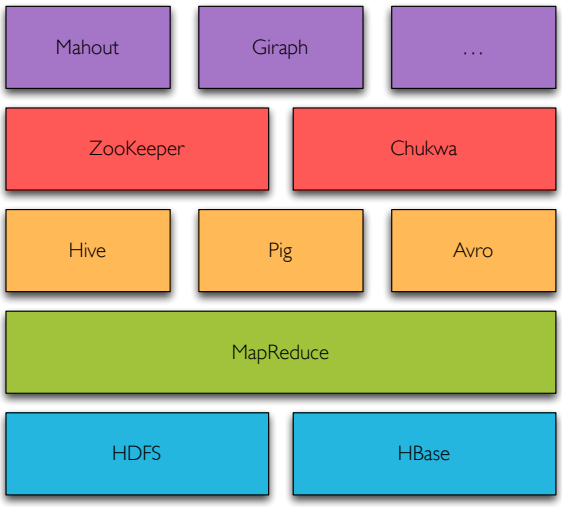


Figure 2.9: The Hadoop Ecosystem. Data storage is performed by HDFS and HBase (see Section 2.4.1). Processing is performed via MapReduce. On top of that are various data access frameworks, such as Hive, Pig (a high-level programming language for MapReduce), or Avro (data serialization). Zookeeper and Chukwa (a logging and management framework) are components responsible for data and cluster management. On a MapReduce library level, Mahout (machine learning) and Giraph (graph-processing) are good examples for the variety of components in the Hadoop ecosystem.

I have outlined the advantages and some disadvantages of the MapReduce approach. The roadmap for Hadoop, however, stipulates the "MapReduce 2.0" framework that will allow other computational data processing methods besides plain map and reduce (Radwan, 2012). This will probably solve some of the architectural problems with some types of algorithms and should make some points easier for specific type of systems.

MapReduce, and especially good algorithm design for the paradigm, is very complex. Not everything could be sufficiently introduced within this introduction. Interested readers can find a good introduction to Hadoop, MapReduce and related technologies in the book "Hadoop – The Definitive Guide" by White (2009).

2.4 NoSQL Databases

While Hadoop and MapReduce are powerful frameworks for sequentially processing big data sets, they lack certain features characteristic for modern database systems. Above all, MapReduce was not intended to serve as a system to replace a database, rather as a tool to process big data in order to be able to store it in a database later. The main problem is that MapReduce and the underlying filesystem were not intended to explicitly handle a random access to particular small data sets. HDFS was intended as a general purpose distributed file system. Soon, developers therefore went on the lookout for solutions that supported these kinds of database operations and were still able to cope with the massive amounts of data MapReduce that was intended for and even use it as a processing tool.

What was basically needed was a database that could serve as the underlying transactional storage for data-intensive applications. Relational databases (Codd, 1970) were optimized for many, but small transactions, such as a banking system. However, when the loads for reading and writing data increase massively, for example for indexing a large number of documents or for web pages with very high demands, these typical tools cannot be used anymore as they lack certain scalability features and were created for single-server use. Relational databases often use a special programming language, called SQL (structured query language) that allows to access and manipulate data in a specifically structured format.

In 1998, the term NoSQL was first used for a database that did not use SQL as a query language (Lith and Mattson, 2010). With the release of a paper about a database that fit in with Google's MapReduce in 2006, called Bigtable (Chang et al., 2006), soon other systems followed that did not offer relational properties, but were distributed systems built from ground up to manage big data. In 2009, a group of developers arranged a meetup whose aim was to discuss open source, distributed, and non relational databases (Eventbrite, 2009). The meetup was called NoSQL, which is a term that is nowadays not anymore used to tell that a database does not offer SQL but instead a movement that emphasizes on distributed high-load systems.

NoSQL is now considered an acronym for *not only SQL* and roughly describes a class of systems that differs from traditional database management systems (DBMS, as described in Section 2.4.4) in many ways. In short, they usually do not require fixed table schemas, avoid join operations of multiple tables and provide built-in *horizontal scalability*, which means the ability to scale out the power of a cluster system through the addition of more nodes instead of more powerful nodes (Lith and Mattson, 2010).

NoSQL databases can be broadly classified into four distinct categories (see Figure 2.10): key-value stores, column-oriented databases, document stores, and graph databases.

Key-value stores are a class of very high-performance databases that allow the storage of all data only as simple key and value objects. Keys are usually sorted and these databases therefore provide extremely fast access to the data.

Column-oriented databases or *bigtables* are also a very high-performance class of database systems, initially invented by Google in 2004 (Chang et al., 2006). The data model can be best described as a sparse, distributed, multi-dimensional, sorted map. At Google it is most often used in conjunction with MapReduce.

Document stores store so-called documents, i.e. any semi-structured content mostly in XML or JSON format, and also enable processing through the MapReduce paradigm. XML (Extensible markup language) and JSON (JavaScript Object Notation) are markup languages[32] for describing documents (text or other) in a structured format. In case of JSON[33], the format is most widely used to serialize data structures.

Finally, *graph databases* are a specialized NoSQL database type that stores graphs with nodes and edges and provides specifically adapted storage and access.

[32] See, for example, http://en.wikipedia.org/wiki/Markup_language
[33] http://www.json.org/, accessed 2012-08

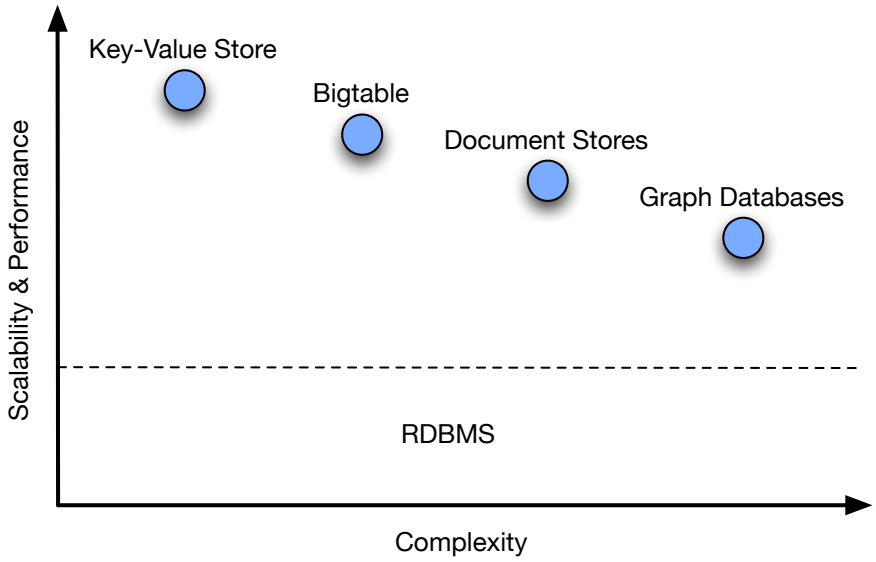


Figure 2.10: Different NoSQL categories: key-value stores, BigTable clones, document stores, and graph databases. For comparison, Relational Database Management Systems (RDBMS) typically have a wide range of different complexity scenarios, but are usually limited in scalability and performance. Adapted from Ivarsson (2010).

Each of these four NoSQL database types has its own advantages and disadvantages depending on the actual use case. They can be mostly discerned by their ability to scale to larger data volumes without loss of performance and the complexity of their data (see Figure 2.10). In a bigger application, different systems can be employed at the same time in order to use a specific NoSQL type for specific problems or use cases. This practice is also called *polyglot persistence* (Leberknight, 2008).

Reliable Database Processing

In traditional relational database systems, the ACID guarantees are widespread key attributes, basically defining properties on the interaction with a database. ACID stands for:

- *Atomicity*: a transaction (or operation on the database) is “all or nothing”, meaning that if an operation is performed it is either performed fully or not at all.
- *Consistency*: any transaction performed on the database must not leave the database in an inconsistent state. Relational database systems usually constrain the data into a fixed schema. Any deviation from that schema would equal a violation of the consistency property.
- *Isolation*: no database operation can interfere with another. This can be best described as two users that concurrently operate on the database. If user A performs an operation, this must not influence operations from user B.
- *Durability*: if an operation was performed, it has to stay that way. In case of an error (software crash, hardware failure, etc.), the changed values have to be changed even after the error was corrected.

NoSQL systems, however, usually do not guarantee all properties of the ACID model, because it makes distribution difficult to impossible. Specifically, the consistency is very hard to achieve. If the database is distributed, only one machine is notified of a new write request. In the time this machine gets the request and forwards it to the other machines, so that all machines are on the same consistency level, the database cannot be considered as consistent (Hoff, 2009a) and would therefore violate ACID, albeit for a short time only.

In 2000, Eric Brewer presented the so-called *CAP-theorem* (Brewer, 2000; Gilbert and Lynch, 2002) that stated: "You can have at most two of these properties for any shared-data system: Consistency, Availability, Partition-tolerance". Consistency is used in the sense as described above. Availability basically means that every request to a system always gets an answer, whereas partition-tolerance describes the ability of a system to recover from failures. The basic idea is that for a distributed, high-performance database system, such as the mentioned NoSQL systems, we usually drop consistency and isolation of the data in favor of availability and performance. This leads to the eventually consistent data model, also called BASE (basically available, soft state, eventually consistent). The idea is that for any change, not all nodes of a parallel system can be informed about the change immediately, but given an enough amount of time, these changes will eventually propagate through the distributed system, making it consistent again. An in-detail comparison between ACID and BASE is given in Table 2.1.

ACID (relational model)	**BASE (NoSQL model)**
atomicity, **c**onsistency, **i**solation, **d**urability	**b**asically **a**vailable, **s**oft state, **e**ventually consistent
strong consistency	weak consistency
isolation	availability first
focus on "commit"	best effort
nested transactions	approximative answers OK
availability ?	aggressive (optimistic)
conservative (pessimistic)	simpler
difficult evolution (schema)	faster, easier evolution
⟵ **spectrum** ⟶	

Table 2.1: ACID vs. BASE database property models. The area between plain ACID and plain BASE is basically a spectrum with many shapes. Some NoSQL databases implement ACID, others do not. Adopted from Brewer (2000).

In the following sections I will introduce examples for the major NoSQL database system classes that were used throughout this thesis. Specifically, I employ Bigtables, document stores and graph databases. Key-value stores, while providing a superior scalability and performance, only allow the storage of completely unstructured data in plain key and value format which is not sufficient for the applications developed in the scope of this thesis.

2.4.1 Column-Oriented Databases: HBase

Column-oriented databases, such as the here described HBase, are a class of database systems that group data, stored as keys and values additionally by columns. Consider, for example, a regular spreadsheet or matrix with rows and columns where each cell is identified by a tuple of a row and a

column identifier. In column-oriented databases, the columns are additionally grouped into so-called *column families*, which means that cells are identified as triples consisting of row, column family, and column.

The idea to this class of database systems was first published by Google-employees (Chang et al., 2006), where they introduced their proprietary database system Bigtable. Before publication it was in development for roughly seven person-years and is in production at Google since 2005. Since 2006, more than sixty internal and published projects of Google are using Bigtable, such as Google Reader, Google Maps, YouTube, or GMail (Hitchcock, 2005).

HBase, in particular, was developed as an open-source column-oriented database for the Hadoop MapReduce ecosystem as a way to gain random access to data, which is as described not possible with plain HDFS. HBase is modeled after Google's Bigtable. It was initially developed by the company Powerset that was acquired by Microsoft in mid 2008 for approximately $100 million (Powerset, 2008). The company, whose aim was to develop a natural language processing search engine for the web, developed a way to store data in a Bigtable-like fashion in Hadoop. They released a first version of HBase in 2007 (Powerset, 2007), a little less than a year after Google published their article on Bigtable.

Data in HBase

HBase stores data as tables, as familiar from other database systems. A table consists of rows that in turn consist of globally defined column families. In these column families key-value objects are stored with a timestamp. Each data value in HBase can therefore be accessed by five identifiers: Table, Row, Column Family, Column, and Timestamp (Taylor, 2010).

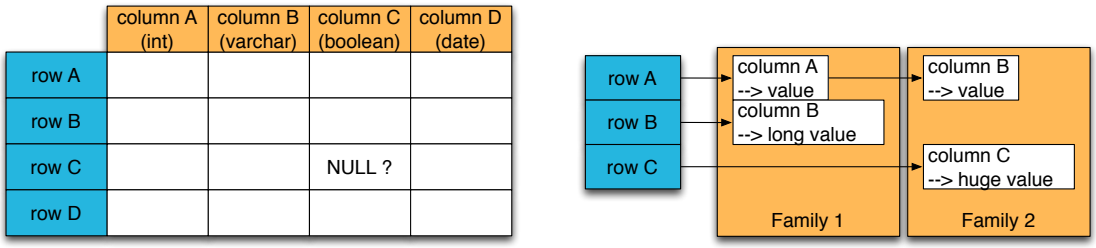
On an architectural level, these column-oriented systems are built such that column values are stored contiguously on disk – in contrast to traditional, row-oriented databases like relational systems, where an entire row of data is usually stored contiguously. The reason behind is that it is assumed that for any particular query typically not all values of a single row are needed but the same type of value for all rows and therefore storage mechanism optimized for this scenario can be used (see Figure 2.11) (George, 2011a).

Transferred to a data structure in a programming language, such as Java, the corresponding theoretical complex for a table in HBase might look something like this:

```
SortedMap<RowKey, List<SortedMap<Column, SortedMap<Timestamp, Value>>>>
```

basically describing a five-dimensional map.

In contrast to traditional relational databases that basically portray a matrix, HBase can be more described as an "adjacency list" that simply stores the entries. It is assumed that a relational table can have many values that are NULL (i.e. contain no data). One big disadvantage of these systems is that every data value has a specific type, such as an integer, for example. With that, a value of NULL even if it does not contain anything still uses up disk space because of the space requirement to denote that this value can be an integer. This is similar to the zeros in a adjacency matrix for graphs. The take up space but contain no useful information which effectively limits the number of columns any particular



(a) Relational databases

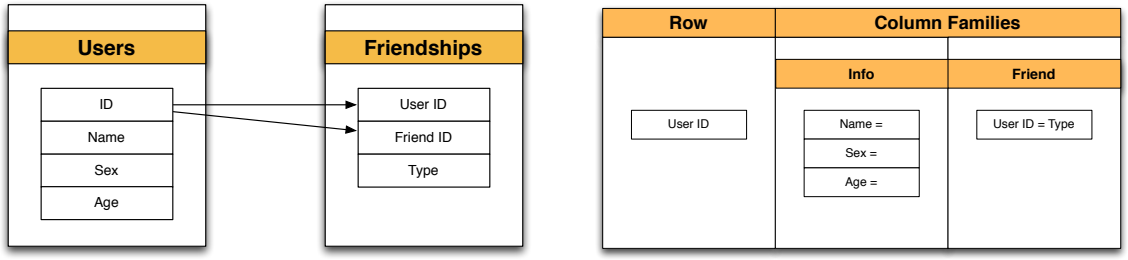
(b) Column-oriented databases

Figure 2.11: Relational vs. column-oriented databases. (a) A relational database is basically a matrix. Cells with no contents (NULL) still take up disk space because of a fixed schema and the definition of a data type. (b) Column-oriented databases, such as HBase, store key-value objects in map form. This implies that "cells" with no contents simply do not exist and therefore do not use any space. Adapted from George (2011a)

table can have. In HBase, which is a map of sorts, these NULL values actually do not take up any space, they simply do not exist. The only schema definition for HBase tables is in the number and names of the column families. Everything else has to be handled by application logic (see Figure 2.11).

Also for the type of data (integers, strings, maps, arrays, etc.), HBase poses no restrictions. Internally, everything is stored as an array of byte values (`byte[]`). Therefore, the logic of an application using HBase has to take care of this as well.

Consider for example a database that contains users and models their friendships. In a relational database the according schema could be implemented as in Figure 2.12(a). It has a table `users` that models every aspect of a certain user, such as the name, sex, age, etc. and a table `friendships` that models the friendships between users by references to the identifiers of the users table. In HBase, on the other hand, the users and their friendships would be modeled within the same table. The row is specified by a user id. In a column family called `info` every information for the user is stored, and in the column family `friendships` we would simply store references to other row ids of the same table (Figure 2.12(b)). This works because HBase stores only the keys and values that are actually set.



(a) Friendship model in relational database (SQL)

(b) Friendship model in HBase

Figure 2.12: Modeling users and friendships in SQL and HBase. (a) Relational databases model schemata with different tables and references between them, such as users and their IDs. (b) In HBase everything is stored within the same table in multiple column families.

Data is accessed in two ways: a Get and a Scan operation. A Get is called with a row id and returns that single row id. A scan defines a starting row id and an end and then basically returns every row

in between. This enables powerful design patterns for the row keys in a table. Instead of having to join various other tables in order to retrieve something, the row key could contain more information than just a simple id and make retrieval therefore easier. In later chapters, some examples for this will be seen. Internally, a Get is the same as a Scan operation but with more limited start and stop keys. Therefore, both operations perform with the same efficiency even though this might seem counter intuitive (George, 2011a).

HBase is built upon Hadoop and stores its data in HDFS. The architecture of HDFS makes it however very difficult to randomly access data, a requirement for database transactions. The Hadoop file system was designed to contain very few[34], very large files that can only be processed sequentially. For a true random-access to data usually millions of tiny files are the norm in more traditional database systems.

HBase bypasses this problem by logically splitting up the data in a table into evenly-sized so-called regions. These regions in turn are made up of HFiles that store datasets of specific column families contiguously. Once a regions' size overflows, the region is split into two. If regions are too small they can also be merged. Merging and splitting is performed by background processes and does not influence regular operation behavior. This techniques allow HBase to create a random-access database on top of a filesystem that only supports sequential reads of data.

A region in HBase is the basic unit of scalability and load-balancing. Contiguous rows are stored together in the same region and different regions are distributed to different servers in the cluster, such that each region is on at least three servers, depending on the replication level of HDFS. This is exemplified in Figure 2.13.

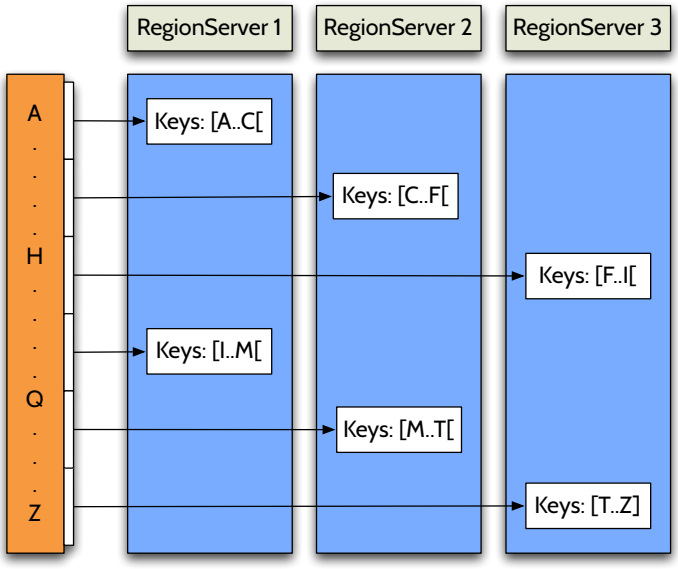


Figure 2.13: Physical layout of region servers. Rows are distributed into region in different servers. Additionally the regions are replicated throughout the cluster (not shown in the illustration). Adapted from George (2011b).

[34] Essentially, the number of files in HDFS is limited by the amount of RAM accessible by the NameNode server process. More RAM means more individual storable files, but also more overhead (George, 2011a).

The regions are distributed across the cluster and replicated through the underlying HDFS. The RegionServers (processes running on the Hadoop platform) manage which region is where and a central master process (HBaseMaster) takes care of initiating region splits and merges (George, 2011a).

ACID or BASE?

If a new request to write something into HBase, called a Put operation, is executed, data is first written into an in-memory store (MemStore) and a write-ahead log (WAL). The WAL is responsible for re-performing changes if a database server failed for some reason. The MemStore writes all incoming requests into memory. At specific points in time, the contents of the MemStore is flushed to disk into the actual HFiles on HDFS (George, 2011a, Chapter 8).

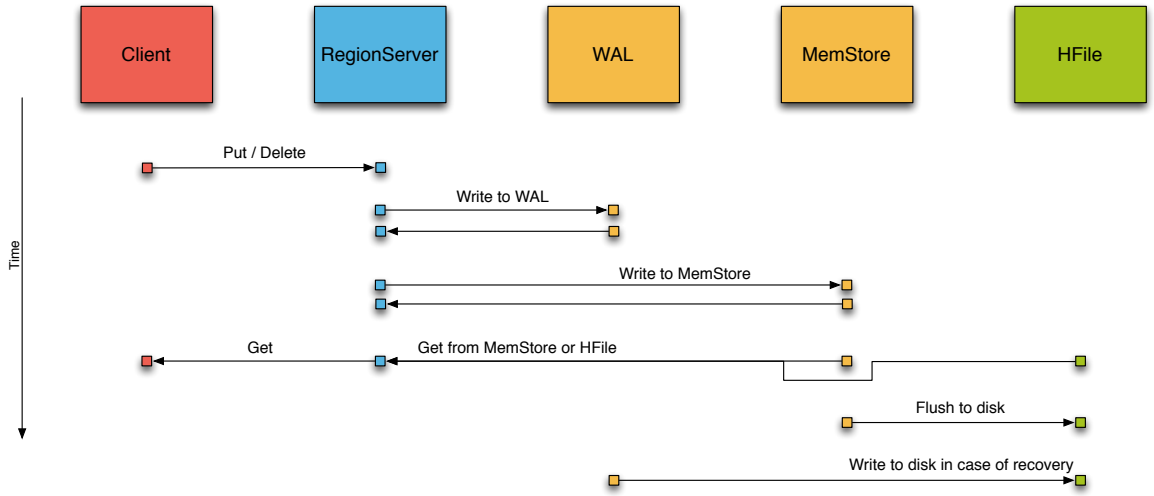


Figure 2.14: HBase write path. A Put operation is first written to the WAL and the MemStore. After a while the MemStore is persisted to disk in form of HFiles. If the RegionServer crashes before flushing to disk, the WAL is used to replay the last changes. Enhanced upon Xiang (2012).

The question now is: Is HBase ACID-compliant or somewhere in between to the BASE model? According to the HBase web site[35], HBase is not fully ACID compliant. Most notably, consistency is achieved eventually (BASE). The biggest difference is within a Scan, as a Scan is not a consistent view to a table, because it does not provide snapshot isolation. This means that when a scan is executed, the single rows are consistently viewed. However, if a row further at the end of the scan is changed between the Scan start and the actual viewing, it is not consistent anymore. Usually, this has no big implications for application design, but has to be kept in mind.

HBase and column-oriented databases are a very interesting type of software. Interested readers can find extensive documentation of all features and properties of HBase in the great book by George (2011a). I unfortunately did not have this book at the beginning of this thesis. It would surely have helped me circumvent some exhausting bug hunts during the early days of getting acquainted with Hadoop and HBase and deploying early versions of the Excerbt text mining system (see Chapter 4). Not

[35] http://hbase.apache.org/acid-semantics.html, accessed 2012-06-29

having the book, on the other hand, proved to be a valuable resource for understanding the internals of Hadoop and HBase.

2.4.2 Distributed Search: ElasticSearch

Searching data is one of the most fundamental processes that led to the development of the Internet as we know it today. Without good search engines, such as Google, Bing, or Yahoo, the Internet could not have evolved. Who want's to build web pages if they cannot be found by anyone?

Searching, however, is not a simple process. The primary technique to employ here is the index. An index is basically a data structure that saves other data in a way that it can be found efficiently by giving keywords. A long-time solution for the creation of indexes is the open source software Lucene[36]. Unfortunately, Lucene is merely a library that can create and query indexes. Configuring and actually using it in a software project can be overwhelming. Also it does not provide anything to distribute the search on multiple computers, which is a very desired capability of nowadays systems where data grows bigger and bigger. More on indices and index creation can be found in Section 4.3.

In order to counteract that, Shay Banon created ElasticSearch[37] (ES). ES is a software framework based on top of Lucene that provides a distributed, near real-time search for large data sets. Horizontal scalability and performance were the primary ideas for the framework. Access to stored data is provided by RESTful web services[38]. In terms of NoSQL category it can be best described as a document-store that is specialized for real-time search in huge data. ElasticSearch's foundation is a sharding mechanism that distributes single Lucene documents onto different machines in a cluster of ElasticSearch servers. It supports on-the-fly shard creation for existing indices as well as the creation of replicas of those shards. A shard is basically a collection of indexed documents. For improved search performance and high availability those shards are replicated throughout the cluster in a way that each shard in the cluster is of approximately the same size and each node gets the same amount of assigned shards.

ElasticSearch can be operated in single-server mode and distributed mode. Each index on ES is specified with a number of shards that the index is split up into and a replication factor for data consistence. In the case of 1 machine, replication more than once is usually not useful, but for a distributed setup vital for data consistency.

Figure 2.15 shows a distributed setup of two indexes in ElasticSearch. Each machine has both indexes. Shards are distributed across the machines. Some of these shards are replicates, depending on the replication factor.

In ElasticSearch everything is basically a node of the cluster. ES discriminates between index holding nodes and non-index holding nodes (clients). Every node can take any request and if it cannot handle the request, it forwards it to a node that can. The implications of this approach mostly affect users that are provided with an easy deployment and integration feature.

[36] http://lucene.apache.org/, accessed 2012-06
[37] http://www.elasticsearch.org
[38] REST (Representational State Transfer) has been first described by Fielding (2000) and can now be considered one of the predominant web services models. Web services are called RESTful, if they operate by the REST principles (Richardson et al., 2007).

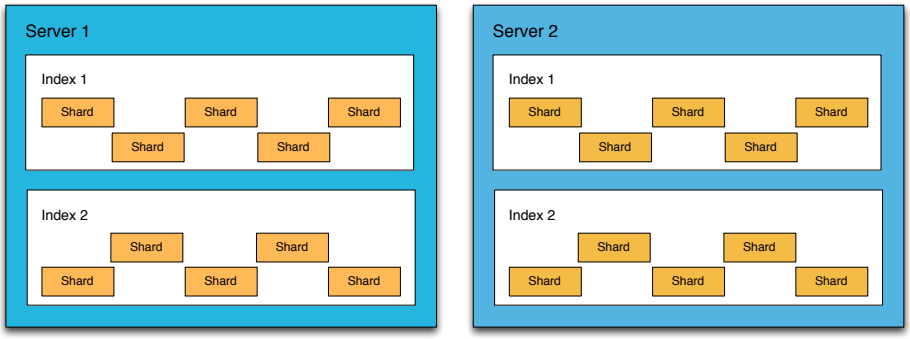


Figure 2.15: ElasticSearch distributed setup. The sample cluster consists of two machines with two indices. Each index has a specific number of shards and a replication level. Shards are distributed evenly across all nodes and replicated on the other machine. A scenario with two servers would equal a simple replication of the whole machine.

Shard replication is done through a mechanism called *push replication*, which means that each document indexed on the cluster is transferred to the destination node before the actual indexing process in order to save network bandwidth. This is essential since in a highly dynamic environment, search speed is not allowed to decrease because servers are occupied with loading new data. Also, the replication is done from the client that initiates the indexing. This leads to faster response times and has the effect that all replicas of a shard are always the same version and do not lag behind, which would be the case when using *pull replication*. However, the overall processing speed is slower because every node has to index everything multiple times and cannot fall back to use finalized index fragments (Banon, 2011).

ES supports a multitude of different query types and modules, a description of which would go beyond the scope of this work. I will discuss only a selection of interesting modules.

Besides regular queries that returns documents corresponding to keyword searches scored by relevance in a paginated way (i.e. first 25 results first, then the next 25), it also supports so-called Scrolls. A scroll is used when all results for a query are desired and the score is unimportant. It is also builds on pagination of the results, but with a constant access time because of the omitted scoring. When scoring documents, the more posterior the result is the longer it takes to be found. This makes it useful for the purposes of the text mining system Excerbt (Chapter 4).

ElasticSearch also provides some interesting other capabilities, such as a River. A river is basically a plugin that listens to pull or push requests to an external resource. Once new data is stored in the external resource, ElasticSearch is informed through the river about the change and can therefore index this at once. This makes new data instantly searchable. Prepared rivers exists for example for the contents of Wikipedia or for Twitter streams.

Another important feature is that multiple indices can be pooled under a common name. This makes it especially useful for distributed index generation. Multiple distributed indexes could be created and then a common alias could be given for all of these indexes. Querying the alias index then would result in the same as if the indices would have been searched individually and results combined.

Documentation for ElasticSearch is hosted at the website[39]. Despite the fact that ElasticSearch only has one main developer, the project is very active with new version updates published at regular intervals throughout the last years.

2.4.3 Graph Databases: Neo4J

NoSQL databases are most of the times no general purpose databases such as relational DBs, but have a very specific use case. In times of information that is highly connected and a mindset that poses "everything is a graph", databases specifically designed for handling network-based data are another main point of interest (Hoff, 2009b).

Neo4j[40] is a high-performance, robust graph database. In contrast to other NoSQL databases, it fully complies to the ACID transactions model *and* provides high availability. As a graph database, it stores data as nodes and edges and has been shown to scale up to billions of nodes (The Neo4j Team, 2012) (see Figure 2.16).

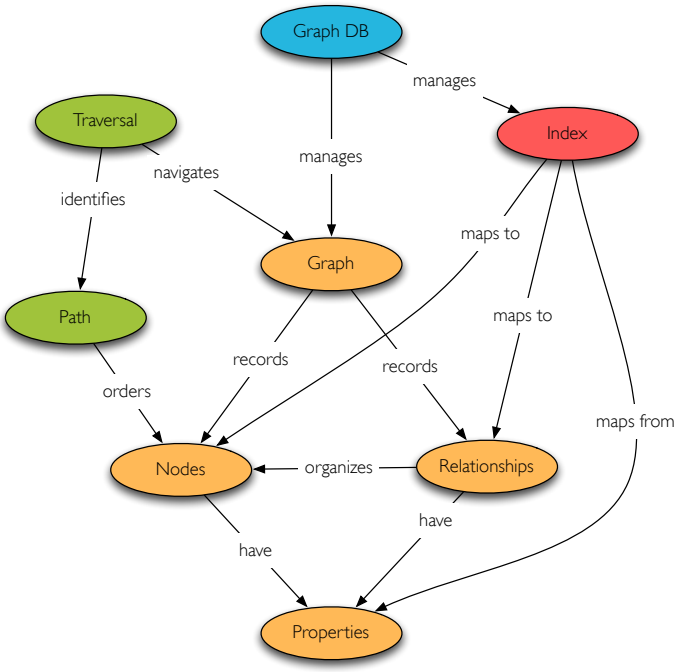


Figure 2.16: Neo4j explained as a graph. A graph consists of nodes and relationships that both can have arbitrary properties (yellow). Additional indexes map keys and values of properties to their respective nodes and relationships for faster lookup (red). The graph database (Neo4j) manages a graph and associated indexes (blue). Client access is enabled through the traversal model (green) that identifies paths to navigate through the graph. Adapted from the Neo4j Website, http://neo4j.org/, accessed 2012-07-23.

Distributed use of Neo4j is possible through sharding, but the graph data makes this process difficult. In a distributed setup, the inter-machine communication should be limited as much as possible. In

[39] http://www.elasticsearch.org/, accessed 2012-06-29
[40] http://neo4j.org/

order to work for network data, graphs have to be partitioned best according to the inherent cluster structure of the network. This can be done with Neo4j but is difficult. Therefore Neo4j is most often used in single-server mode, but it is very powerful at that. It can handle graphs up to billions of nodes on decent single machines (Kollegger, 2011).

To query the database, Neo4j employs the so-called *traversal* model, meaning a query starts at a specific node and subsequently traverses along the edges to aggregate the queried values. Neo4j has a built-in query language, called *Cypher*. A sample Cypher query might look like this:

```
START john=node:node_auto_index(name = 'John')
MATCH john-[:friend]->()-[:friend]->fof
RETURN john, fof
```

which starts at a node with the name "John" and returns all friends of John's friends, i.e. the neighbor's neighbors of the node John, but not his direct friends (The Neo4j Team, 2012).

Basically, Neo4j and Cypher have been built to provide an efficient database solution for highly connected data compared to SQL (relational databases) that is made for data that has a limited number of connections (Taylor, 2011).

Additionally to the traversal querying, Neo4j contains indexes (again based on Lucene) to find specific nodes and edges by means of specific attributes instead of an identifier which enables a fast finding of relevant nodes and then starting from there with a query.

2.4.4 Wrap Up

The NoSQL movement has a lot of different systems now ready to use. Each of the system obviously has another speciality. Column-oriented database systems perform well with semi-structured schemas as the only thing to specify are the column families. Document stores and search solutions such as ElasticSearch have their niche in storing basically schema-free or enable fast and efficient search on the data. The purpose of graph databases is to efficiently store and traverse highly connected data in any form.

A comparison with relational database systems is basically an apples vs. oranges comparison. While being built for different reasons and with different backgrounds, all type of systems have their qualifications in different areas.

While the NoSQL movement was duped "say no to SQL" at the beginning, it now became much broader and does not necessarily restrict databases not to use any type of SQL. As a query language SQL is a quasi-standard for databases and as such it is still highly desirable to have a similar language to interact with data, even if some constructs might not be possible in certain types of NoSQL databases.

Generally, one can say that the employment of a particular database is highly dependent on the complexity of the data and the performance that is needed for the system. Figure 2.17 shows some examples for trade-offs between data complexity and performance compared to what relational databases management systems (RDBMS) are capable of providing. The more complex the data gets, the more the performance of such systems drop. Even though the performance requirements for

applications that perform semantic processing (such as text mining) are lower than for example for a salary list application at a company, RDBMS still scale too little and the discrepancy between desired and actual performance is too high. Here, NoSQL data stores have their justification.

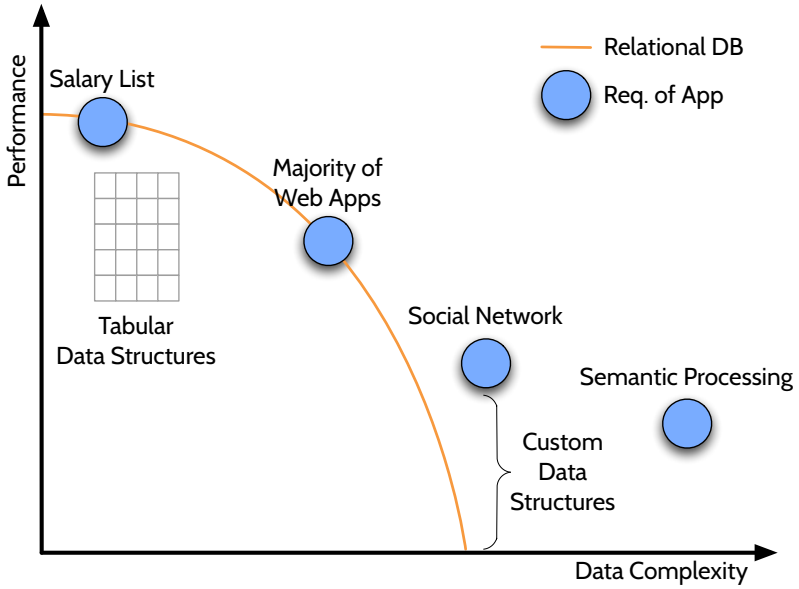


Figure 2.17: Performance capabilities of relational databases vs. actual requirements of applications. When data complexity for the application increases, the performance of relational database management system drops constantly below a point of intolerability. Adapted from Ivarsson (2010).

An interesting view on relational and NoSQL databases can also be given by investigating their power to perform online analytical processing (OLAP) or online transaction processing (OLTP). While OLTP is a paradigm for applications on databases that perform without delays, OLAP is a technique to aggregate, process, and analyze data. Figure 2.18 shows a selection of data processing systems and their place in the spectrum between OLAP and OLTP. As another dimension, the affinity to either general purpose or highly complex data (graph affinity) is given.

All in all, NoSQL databases are made for storing and retrieving huge amounts of data efficiently. Whether the data is connected, structured or completely schema-less does not matter at first. This is where the existing variety of systems comes in handy.

For this thesis I have employed all of the presented systems in some or other ways (polyglot persistence). Each have their unique use case for the data in Excerbt which is explained in Chapter 4 in more detail.

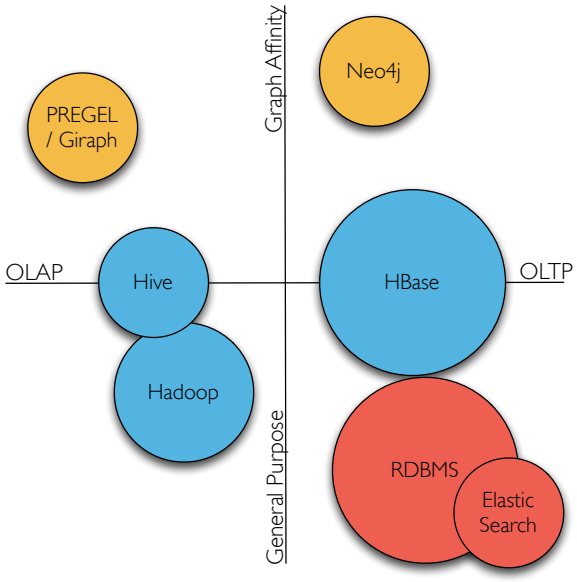


Figure 2.18: OLAP (online analytical processing) vs OLTP (online transaction processing) with regards to different graph-based and non-graph-based data management systems. Different data management systems are depicted as green bars. The width of the bar denotes the range of perfect applicability of the corresponding system. Created with influences from Webber (2011).

2.5 Natural Language Processing

> "The ultimate goal of research on Natural Language Processing is to [...] understand language."
>
> (*Manning and Schuetze, 1999*)

Natural Language Processing (NLP) has its history as early as the 1950s, when Alan Turing published his work on "Computing Machinery and Intelligence" (Turing, 1950). In this paper, he proposed the Turing test[41] that was designed to test computers for intelligent behavior in oral-like communications. His studies led to the development of many systems that tried to simulate human talk behavior, such as ELIZA (Weizenbaum, 1966) or Jabberwacky[42], just to mention a few. But only in the last decade or so, NLP has come to a point where huge amounts of texts are readily available for analysis and as training data to machine learning approaches. NLP can now be considered an industry reaching millions of people (Jurafsky and Martin, 2009), mainly due to the huge influence of the Internet on people's lives.

The basic aim of NLP and (computational) linguistics in general is to understand how humans communicate with each other through verbal or written messages. In computationally aided linguistics, the focus is on the automatic processing of human natural language. This involves mainly statistical analyses of language and the creation of heuristic language models that can operate upon that statistical information (Manning and Schuetze, 1999).

[41] http://en.wikipedia.org/wiki/Turing_test, accessed 2012-06-08
[42] http://www.jabberwacky.com, accessed 2012-06

This section will cover the most important concepts of NLP. In the context of this thesis, I use the term NLP for individual processes and techniques in order to bring structure into unstructured textual information and the term Text Mining (TM) (see Chapter 3) for an application built upon those techniques in order to extract knowledge from texts.

2.5.1 Information Retrieval

Up to the 1990s, people preferred getting information from other people instead of using the Internet with sophisticated information retrieval methods (Chi, 2012; Manning et al., 2008). The problem probably can be attributed to two points: a limited availability of web resources that could be retrieved (the Internet was still in its infancies at that time) and limited possibilities for the retrieval process because of missing technology. The largest obstacle back then was, who to ask for information. Today, search engines such as Google or Bing are the first places to go when looking for information. But only in the mid-90s, when companies such as Yahoo (Yahoo, 2005) or Google (Google) were founded, the important technological foundation for information retrieval and the basis for the modern Internet were laid. But even though information retrieval only became famous in the last two decades, it has been around for about 50 to 60 years (Broder, 2006).

Manning et al. (2008) defines the basic building block of a search engine, information retrieval (IR), as:

> "Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)."

Translated to a more colloquial definition, it states that IR is the process of finding the documents a user is interested in. In their book, Manning et al. (2008) provide an overview on various methods that perform information retrieval tasks and how to store data accordingly in order to find it efficiently. Basically, the ground work for an IR engine is done by a program that converts unstructured text data into a structure optimized for searching, a so called index (see also Section 2.4.2).

IR can be basically distinguished into two methods: retrieving documents that correspond to a certain query and retrieving documents that are similar to other documents (see Figure 2.19).

Query-based IR

In query-based IR, the main point is to find documents that correspond to certain criteria as specified by a query. A query could, for example, entail keywords that have to appear in the document, or other metadata associated with it. There exist various approaches through term vocabulary indexes, where each word in the documents is indexed with a reference to the original document. A user query for specific keywords then has to simply query the index and display the documents. ElasticSearch (see Section 2.4.2), for example, could be used to create such an index. Other approaches, such as probabilistic methods, where for example a Bayesian classifier is trained to distinguish between relevant and irrelevant documents according to a query (Manning et al., 2008, Chapter 11) are also conceivable.

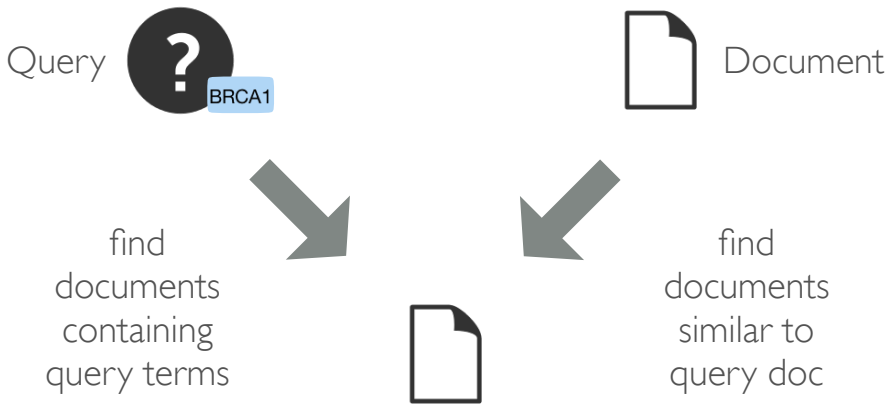


Figure 2.19: Information retrieval methods: query and document-similarity.

Examples for query-based IR include popular search engines such as Google or the Pubmed search where one can search for biomedical publications

Similarity-based IR

Similarity-based IR works similar to the probabilistic methods from above. The main difference is that here documents are pre-clustered according to some similarity property to each other and not to particular relevance criteria. When a new query is submitted, it contains a document whose similarity to other documents has to be determined. Many clustering algorithms can be used for this task, an extensive overview is, for example, given in Andrews and Fox (2007).

Examples here include the Pubmed related articles feature. When an article is viewed, on the right side a list of related citations is displayed (see Figure 2.20).

Zipf's Law and the Curse of Stop Words

In order to better understand, how indexing and document clustering work, a useful measure is to analyze the frequency distribution of words across documents. In Figure 2.21(a) the frequencies of the highest occurring words of plain English texts were plotted. From the shape of the curve can be deducted that these frequencies follow a specific distribution. This frequency distribution is called *Zipf's law* and means that the second most frequent term has half the number of occurrences than the most frequent term and so on. This means that the frequency of term cf_i is proportional to $1/i$ (Zipf, 1932), where i is the position in the sorted frequency list:

$$\mathrm{cf}_i \propto \frac{1}{i}$$

Although the Figure was created for the distribution of words in the English language, Zipf's law does also hold true for essentially any language, even ones that are randomly created by monkeys typing blindly on a keyboard (Anderson, 2006). The reason is that in a random language, letters and space

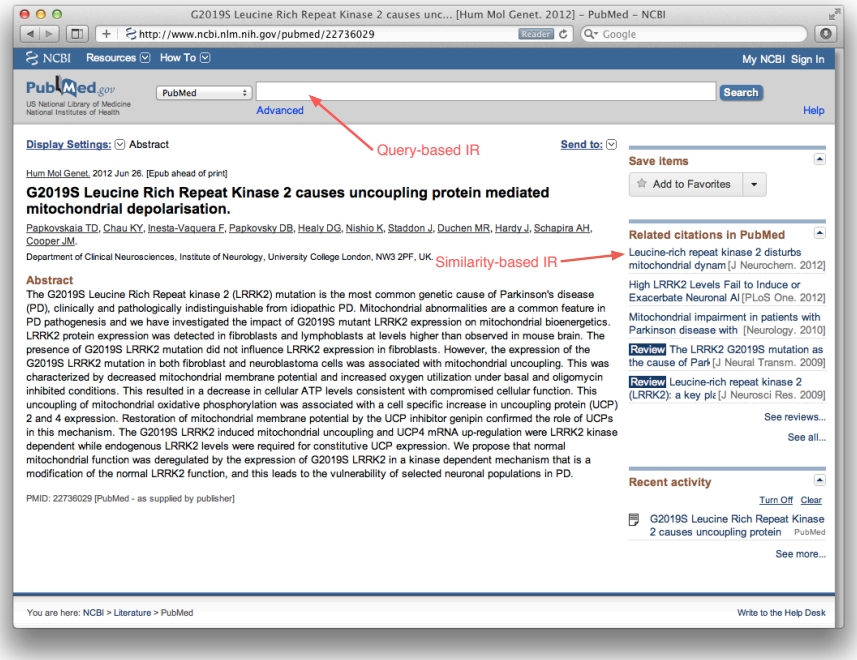


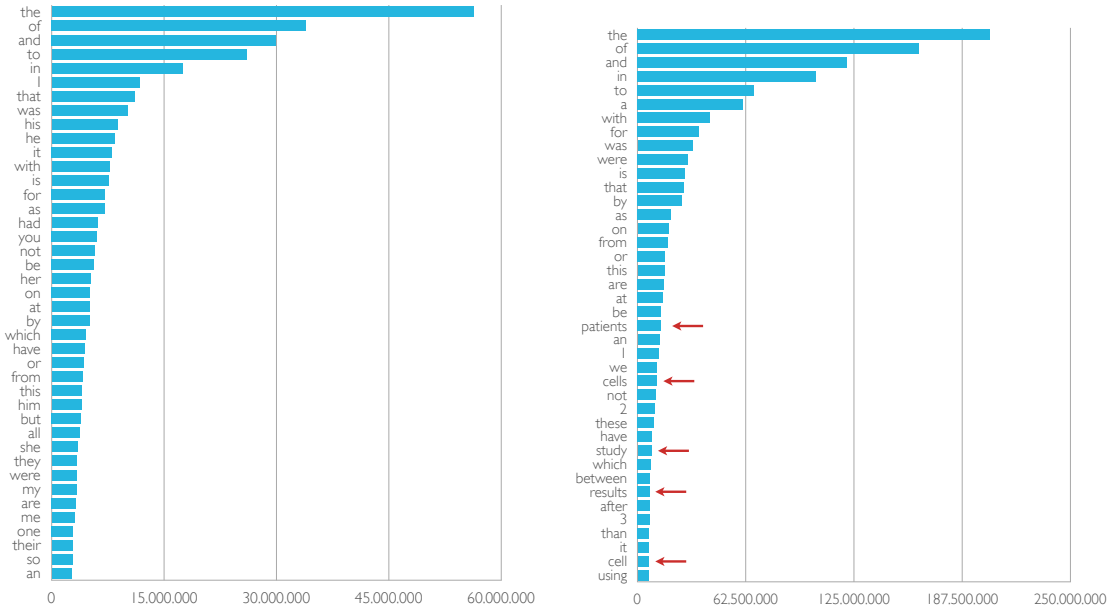
Figure 2.20: Pubmed Search: query-based and similarity-based. Screenshot obtained 2012-06-29.

characters all have the same probability, which means that words consisting of only one letter are the most common and the frequency for any word decreases with its length. In natural languages, obviously, the word frequencies are not dependent on the length of the word. This basically explains that Zipf's law for text is not a linguistic feature of a particular language but simply a statistical property that has its basics in mathematics.

Rebholz-Schuhmann et al. (2005) conducted a study for the frequency distribution in biomedical publications. I have recreated the frequency distribution for all texts in Pubmed and Pubmed Central. The word frequency distribution still follows Zipf's law, though it is interesting to note that the words are not the same than for plain English (Figure 2.21(b)). While the top most occurring terms in plain English are mostly determiners and similar syntactical words, in biomedicine even nouns such as patients, cells, study, or results are within the top 50 most frequent terms, refuting the earlier statement that those words are most often filler words.

The concept of Zipf's law and the construction of languages by formal notations has also been applied to genomic DNA sequences. Searls (2002), for example, compared DNA to a formal language and the genome of a life form to a book written in that language. Even then the distribution of different parts of the genome seem to follow Zipf's law, which makes it a universal distribution not only valid for texts but also other statistical phenomena.

Zipf's frequency distribution provides the basis for statistical analysis of texts. These most frequent terms occurring in a language are also called *stop words* and pose a big problem to indexing-based information retrieval solutions. Because their frequency is extremely high, for example for the word



(a) English

(b) Biomedicine

Figure 2.21: Stop words in standard English and biomedical literature. (a) English stop words. Data was obtained from the Wictionary frequency list (http://en.wiktionary.org/wiki/Wiktionary:Frequency_lists/PG/2006/04/1-10000) and the top entries are displayed. (b) Biomedical literature stop words. Domain-specific terminology in biomedicine has been highlighted red. Data was obtained by analysis of Excerbt's data corpus (see Chapters 4 and 5) as at June 2012. I performed a hive query on the sentences that estimates n-gram frequencies in text data. The query took 5 minutes. Note that the values are not directly comparable, but the overall shape of the distributions are.

"the", they occur basically in every sentence in every document, creating a huge result list when queried for that is of no inherent use to any user. When looking at the type of words stop words in most cases are, it becomes evident that these are mostly filler words that clutter text and have limited meaning but make sound the language nicer. Search engines such as Google or Bing[43] therefore ignore these stop words in the indexing and in the querying steps, thus limiting all the words that can be searched to the words that have enough "meaning" attached to them.

2.5.2 Named Entity Recognition

An important task in natural language processing is the identification of words in text that describe specific concepts, such as persons, companies, places, genes, or diseases. Named Entity Recognition (NER) is the process of identifying these concepts, the *named entities*. There are various approaches to NER, the simplest one being that of a *dictionary-based* approach. Unfortunately, this approach is only simple in terms of the idea. The implementation poses significant obstacles as evidenced in Section 4.3. In this approach, a lexicon of some kind (for example a list of diseases or a list of company names) is provided and searched in text data by traditional methods and indexing. Other approaches to use statistical and *heuristic* methods. Here, the methods try to assign words in a sentence to a specific type of entity, such as a gene. Algorithms range from simple Bayesian classifiers to Hidden Markov Models and similar approaches. The main disadvantage here is that they need a lot of expensive to get training data in order to have a large enough set to later perform well. A third approach is *grammar-based*, where linguistic knowledge is included in the recognition task. Unfortunately, this approach needs much work in form of annotating resources by computational linguists in order to define grammar-based approaches (Jurafsky and Martin, 2009).

All in common is a need for expensive training data in one or the other form. In biology we have the advantage of an abundance of resources for biomedical entity names, so called ontologies (see Chapter 4). Obviously, any resource cannot keep constantly and immediately up-to-date, once new entities have been named. While other approaches might detect new entities, it is within the acceptable bounds of a well-performing system not to have perfect sensitivity.

In this context, the relations between concepts or entities also play a vital role. Concepts can be synonym, homonym, or antonym to each other. Depending on the use-case knowledge about these relations might be important. *Synonym* means that two different words mean the same or something similar. *Homonym* are words that sound similar, but are completely different in meaning, such as "red" and "read", or "to, two, too". *Antonyms* finally are words that have the opposite meaning, such as "love" and "hate". Such relations between words should be represented by an ontology, which can be described as "specification of representational vocabulary" (Gruber, 1993). Put differently, an ontology is a resource that defines the common language for specific systems through concepts or entities and their relations to each other.

In biomedicine, as exemplified in Figure 2.22, named entity recognition tries to detect concepts with a specific meaning and a name. In this example, the sentence has three entities annotated: ERF1, ethylene, and jasmonate. In a dictionary-based approach, such as the one used here, the concepts can

[43] http://www.bing.com, accessed 2012

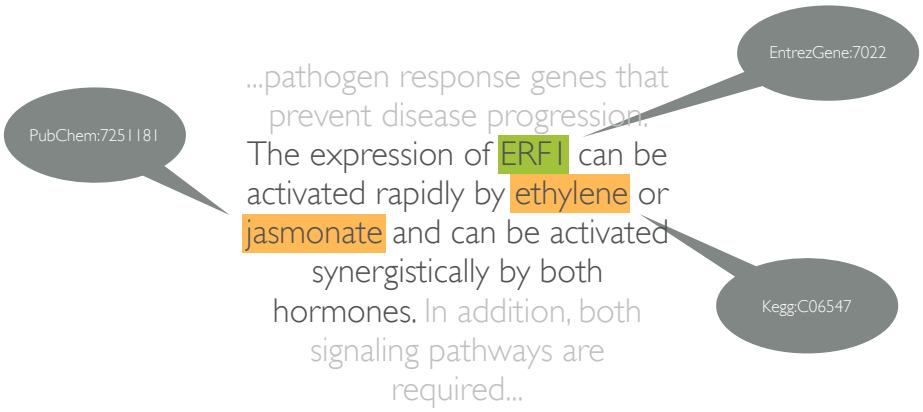


Figure 2.22: Named Entity Recognition. Named entities, such as genes, metabolites, names, places, etc. are identified in free text and connected to databases.

be directly mapped to a lexicon. In this example ERF1 is mapped to the EntrezGene database, ethylene is mapped to the Kegg Drug database and jasmonate is mapped to a lexicon called PubChem.

With a non-dictionary based approach, the mapping to ontologies could also be performed. However, for words that are not in ontologies, the mapping fails and does not give any useful information.

2.5.3 Part-of-Speech Tagging

Humans intuitively know the meaning and type of words within sentences. Computers, on the other hand, need to have standardized processes in order to identify the inherent syntax and semantics of sentences. One such approach is Part-of-Speech tagging, a process where for each word in a sentence, the grammatical type (the part of speech) of the word is determined. In other words, words are assigned whether they are nouns, verbs, adjectives, determiners, etc. (Manning and Schuetze, 1999). An example for such a tagging can be seen in Figure 2.23. Here a sample sentence was taken and each word was tagged with the corresponding part-of-speech.

In addition to the tags, I have highlighted nouns, adjectives, adverbs and verbs. In the example sentence it is clearly visible that these types of words are more important in a sentence than the non-highlighted words, which mostly consist of stop words that are of little value for the understanding of a sentence.

In order to work, most POS tagger need labelled training data, for example the Penn Treebank corpus of the English language (Marcus et al., 1993), which provides a standardized set that most POS tagger rely on.

There are different approaches to Part-of-Speech tagging that use machine learning techniques on those training corpora. Toutanova and Manning (2000) present a part-of-speech tagging approach based on a maximum entropy model, which is known as the Stanford POS Tagger[44]. In 2003, they enhance their system by including a cyclic dependency network (Toutanova et al., 2003).

[44] http://nlp.stanford.edu/software/tagger.shtml, accessed 2012-06-14

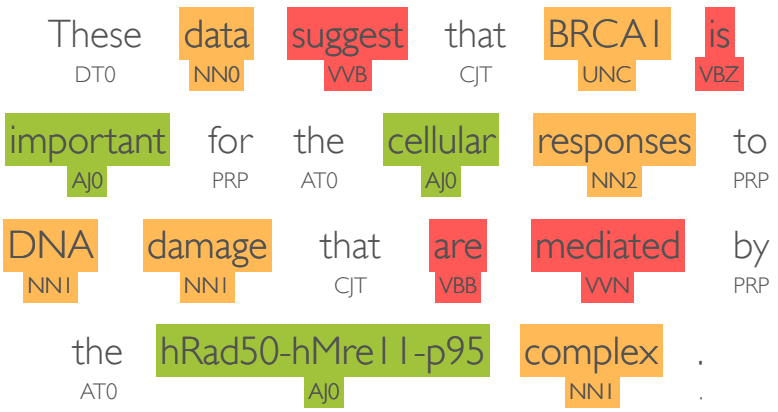


Figure 2.23: Part-of-Speech tagging. Each word in a sentence is labelled according to the semantic word type, e.g. nouns, verbs, determiners, adjectives, adverbs, etc. For a complete list of all POS tags, see the table on page 154. In this sentence, words of importance are highlighted. Nouns orange, adjectives green and verbs red. This tagging was done with the Stanford POS tagger (Toutanova and Manning, 2000). Note that the word BRCA1 was wrongly tagged as UNC, meaning that the tagger did not correctly recognize the correct type of a noun for the gene/protein name.

An in part unsupervised POS tagging method with a very special focus was published by Das and Petrov (2011). They use an approach to label data in a language that has no training data resources but have a lot of translations from another language that has training data available. Basically, they have training data in one language, e.g. English, and transfer this labelled data via a graph-based label approach onto another language where they have translated texts.

Manning and Schuetze (1999) and Jurafsky and Martin (2009, Chapter 5) give further information on the topic. A list of POS tags, as used by the Penn Treebank and throughout the remainder of this thesis, is given in the Table A.1 (page 154).

2.5.4 Semantic Role Labeling

The ultimate goal in natural language processing is for a computer to be able to work with text in a way that humans can relate to. While the previously mentioned techniques are a big help in accomplishing this task, they do not yet provide a unified view on the semantic structure of single sentences. The basic question, when interpreting any text, is to find out *who* did *what* to *whom*, *when*, *where*, and *how*. One approach to answer these questions is to identify the verbs in sentences and subsequently assign the semantic relationship that different parts of a sentence adopt with the verb. Typical such semantic roles are the agent, patient or instrument of an action. Additional roles also include locations, time designations, manners or causes (for a full list of semantic roles and their explanation see the Table A.2, page 155) (Marquez et al., 2008).

Semantic Role Labeling (SRL) is the task of assigning these semantic roles onto parts of a sentence. It has its history in the FrameNet[45] project and a first automatic system to work with that data was developed

[45] https://framenet.icsi.berkeley.edu/fndrupal/, accessed 2012-06-15

by Gildea and Jurafsky (2002). In 2005, the PropBank projcect[46] was introduced as a collection of sentences annotated with semantic roles in addition to the Penn TreeBank (Palmer et al., 2005). In the same year, a first introduction of the SRL concept to the biomedical community was made by Kogan et al. (2005) where they discovered that a system trained on PropBank could also perform well on biomedical data. Meanwhile, more semantic role labelers were developed with a general approach (Pradhan and Ward, 2008; Punyakanok et al., 2008; Toutanova et al., 2008) and also with a specialized labeling approach within the biomedical area. Tsai et al. (2007), for example, developed the BIOSMILE engine that worked specifically with biomedical data and Bethard et al. (2008) implemented a SRL engine that was specifically designed to correctly identify protein transport predicates through focusing on the roles agent, patient, origin, and destination. Later, also systems that dealt with the adaptation of the PropBank corpus, which was a corpus of labeled newspaper articles, to the biomedical domain were published (Dahlmeier and Ng, 2010).

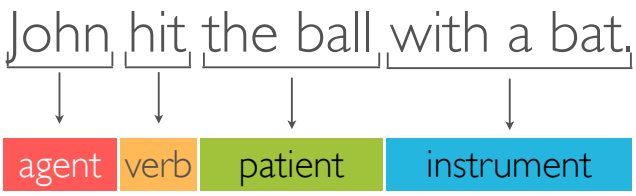


Figure 2.24: Semantic role labeling. Semantic arguments are determined according to their verb. The verb *hit* is specified by an agent *John* (the executioner of an action), a patient *ball* (the receiver of an action) and an additional instrument *bat*.

Figure 2.24 shows a sample semantic role labeling. The action in this example is that something is hit. John is the agent of the action (the doer) and the ball is the patient (the receiver). The bat could be seen as an instrument for this kind of action.

Between a verb and the arguments in a sentence a semantic link is formed because of the meaning of the verb. A verb might have only an agent and a patient, but can have up to six other different roles, such as an experiences, a theme, an origin, or a destination. For example the verb “walk” can have an agent or a destination (where). An agent argument therefore is something or someone that actively performs an action. A cause argument on the other hand is something that involuntarily causes an action or situation. For example, in the sentence “Rain makes me happy.”, rain is the cause of happiness, but does not actively do so (Palmer et al., 2010). A list of various semantic roles can be found in Table A.2 (page 155). The output of a SRL engine is called a predicate-argument structure. Every such structure contains exactly one verb and the various attached roles that depend on that verb, which is called a predicate-argument structure (PAS).

In the following, I will briefly introduce two select semantic role labeling systems: SENNA and ShallowSRL. In the context of the Excerbt text mining system (see Chapter 4), Barnickel et al. (2009) introduced the proprietary SRL engine SENNA (Collobert and Weston, 2007) at large to the biomedical domain. It was also used largely throughout my work but has been replaced at the by ShallowSRL, a system implemented at our institute (the Institute for Bioinformatics and Systems Biology at the

[46] http://verbs.colorado.edu/~mpalmer/projects/ace.html, accessed 2012-06-15

Helmholtz Zentrum München) for newer versions. An in-depth introduction into semantic role labeling can be found in the book by Palmer et al. (2010).

SENNA – A Neural Network Architecture for Semantic Extraction

SENNA is a semantic role labeling engine developed by Collobert and Weston (2007) at NEC labs. Their aim was to improve upon existing SRL technologies that were very slow performing because of the use of syntactic parsers[47]. According to their analysis, their system performed with an 122-fold speed increase compared to another system by Punyakanok et al. (2008) with an identical test set. Instead they built a SRL engine that maps roles directly onto a sentence without any intermediate steps, thus performing computationally very efficient. They achieved high performance values, on average about 0.2 seconds for the labeling of a sentence compared to a couple of seconds that other approaches take, by implementing a multi-layer perceptron neural network architecture. SENNA achieved an accuracy performance of about 85 % depending on the test, which was comparable to other state-of-the-art SRL systems at the time.

Without going into details of the implementation, the system has been shown to work in a large-scale environment by Barnickel et al. (2009). It basically uses an approach where each word in a sentence is labelled separately by a trained neural networks and later post-processed into predicate-argument structures. It thus omits expensive syntactical parsing steps that were employed by other approaches, resulting in an increased speed.

SENNA was initially written in Lua, a scripting language often used for video games[48]. This, however, had some disadvantages for maintainability and usability. Later the system was completely rewritten in C[49] but still with the same basic idea behind (Collobert et al., 2011). The newer C version (SENNA 2.0) was used for this work.

Unfortunately, the SENNA engine has some evident problems for the application in a general purpose text mining suite. First of all, the technology is proprietary and patented in the United Stated (Collobert and Weston, 2008) and therefore an applicability is questionable. While this usually does not count as a reason in academia, the consequence is that the authors do not offer the source code or deeper explanations to their training method. For this reason, several options to train and enhance the system with personalized methods and own data within our domain are unavailable.

ShallowSRL

For the above stated reasons, we decided at our institute to develop a new semantic role labeling approach, called ShallowSRL. This engine will not be described in exhaustive detail since it is the work of a colleague of mine, Philipp Blohm, but a short introduction is given since Excerbt switched to this labeler within the last year.

[47] Syntactic Parsing is a form of analysis of a sentence in order to assign syntactic properties to words. The output usually is a dependency tree that has the single words as leafs and their role within the sentence. Syntactic parsing is a time-consuming process (Manning and Schuetze, 1999).

[48] http://en.wikipedia.org/wiki/Lua_(programming_language), accessed 2012-06-18

[49] http://en.wikipedia.org/wiki/C_(programming_language), accessed 2012-06-18

Figure 2.25: The Shallow SRL approach. Adapted from personal communications with Philipp Blohm, 2011.

ShallowSRL is built in a similar way as SENNA. But instead of using a neural network approach, it employs a support vector machine (SVM) based approach that is implemented in a step-wise process as outlined in Figure 2.25. In a first step, sentences are fragmented according to special keywords (such as and and or) and punctuation marks. After that, these fragments are chunked in order to merge common word groups into single groups, e.g. articles, adjectives, and nouns. Role labeling is then performed for each chunk, and each word in the same chunk is annotated with the same role. An exception to this is the verb chunk which can contain modifier and negation arguments, such as “can”, “may”, or “not”, which are corrected in a post-processing step. Based on the tags within the single phrases, predicates are extracted and supported with modifiers. Also, instead of performing the labeling on each word separately, it uses a chunk-based window approach.

ShallowSRL currently achieves an accuracy of 76.6%, precision of 74.4%, recall of 76.6% and an f-measure of 75.5% (Blohm, 2014) based on chunks within considered clauses on the PropBank data set (Palmer et al., 2005). ShallowSRL is still under heavy development by Philipp Blohm, therefore these values might change in the near future. Evaluations show that the performance is comparable to SENNA (in terms of quality and speed) but without the aforementioned disadvantage of an unavailability of training resources.Other advancements could, for example, include the resolution of hierarchical role labels. Consider a main clause and a subordinate clause. The verb of the subordinate clause would be dependent on the verb of the main clause therefore giving a hierarchical relation between extracted predicate-argument structures instead of plain flat ones.

2.5.5 N-Grams

Text can be described and analyzed on many levels. Ranging from individual words to sentences, paragraphs, or entire documents at once, a fifth structure is imaginable, so-called *N-Grams*.

N-Grams are n-tuples of items in consecutive order of appearance in a greater pool. This is a very general definition, but in our case we can restrict it to words in order in a text document. An unigram ($n = 1$) is therefore each individual word. A bigram is each two consecutive words within a text and so on. An example for various n-grams is given in Figure 2.26.

In NLP, the use of N-Grams is most widespread in the area of machine translation and automated text generation. N-Grams and their frequencies in naturally occurring texts provide the statistical fundaments for probabilities on phrasing and likely translations between languages (Jurafsky and Martin, 2009). The more likely an N-Gram with a higher n that still fits a translation of individual words is, the more likely it provides the correct translation. Other use cases of N-Grams might include a frequency database for improving typing and auto-correction on keyboards for mobile phones (Klarlund and Riley, 2003; Tarasewich, 2007).

N-Grams provide an interesting method to gain statistical insights into the behavior of language. As demonstrated later in this work (Section 4.6), n-grams not only allow a use in machine translation but can also give interesting analytical insights on the development of languages.

Apart from this selection of techniques, NLP consists of much more. Good introductions provide the books Manning and Schuetze (1999) and Jurafsky and Martin (2009), that are both considered the

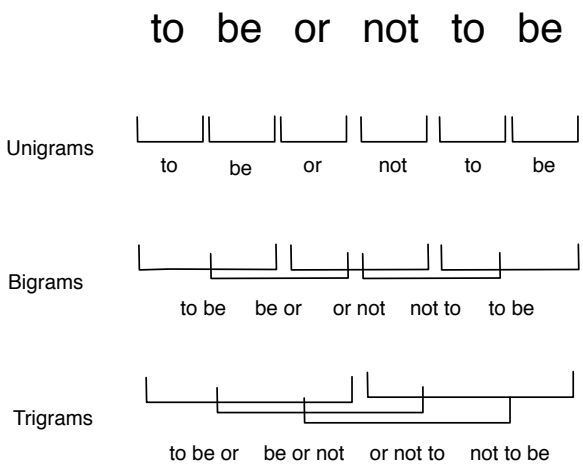


Figure 2.26: N-Grams. The words in the sentence "to be or not to be" in contiguous order of 1 (uni), 2 (bi), and 3 (tri) tuples each.

"bibles" of computational linguistics. The former focuses primarily on word and phrase frequencies and builds techniques upon them, the latter gives more of a general introduction to language and speech processing.

2.6 Topic Maps and Semantic Knowledge Representation

At some point when developing a system that extracts knowledge from unstructured data, one must consider how to store the retrieved data, i.e. how to store (human) knowledge in a useful way on a computer. The basic idea in the area of knowledge representation is how to store any knowledge in such a way that also a computer can achieve intelligent behavior on the data by, for example, inferring deductions (Nenova, 2008).

Most of the computer systems available today, such as search engines or the files on a computer, store documents. When we ask Google for something particular, for example "Which lenses fit best to camera XYZ?", search engines interpret this question as a list of keywords and basically return documents that contain those keywords not necessarily in that order. The answers that we're looking for might be written in some of these documents, but not in a way that a computer can present us with a single *answer* instead of a multitude of unstructured text documents where the answer might be contained. Put differently, current search engines cannot resolve the semantic fit between the original question and a (hidden) answer within a document.

In 1945, Vannevar Bush, an American engineer and one of the most influential advisors in World War II, published an article called "As We May Think" (Bush, 1945). In his article, he invented the concept of *associative thinking*, in which humans brains operate by association:

> "With one item in its grasp, it [the brain] snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain."

Bush (1945) basically calls it “selection by association, not by indexing”. With our current systems, computers “think” in terms of documents and links between them. If a computer would think in form of facts and links between those facts (and additional links to the documents that contain these facts), computers could, similar to a human, deductively reason upon that information and autonomously draw conclusions.

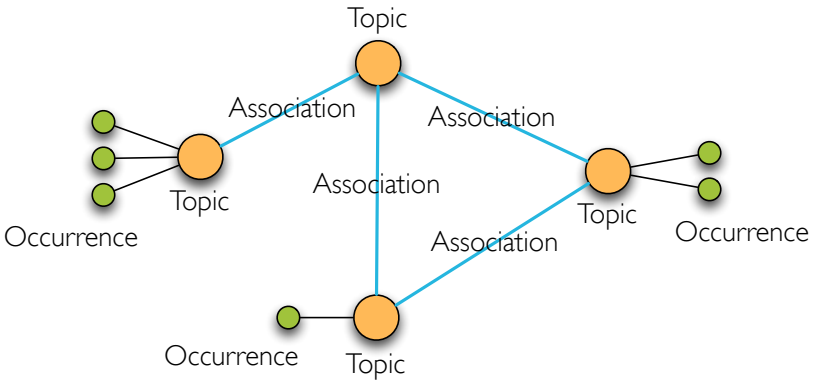


Figure 2.27: A simple topic map. Topics are basically networks consisting of two different node types: topics (orange) and occurrences (green). Topics can be connected to each other through associations.

Topic Maps attempt to provide a data standard for the representation of knowledge in associative form. The concept was originally developed as a book-of-the-book index, an index that allowed the merging of indices in books, but soon generalized into the concept as described here (Ahmed and Moore, 2005).

A topic map basically consists of *topics* that represent any concept, such as persons, companies, genes, or diseases. These topics can be connected to each other via *associations*. Topics and associations basically form a hypergraph since an association can have more than two topics partaking. Additionally, topics can have *occurrences* that describe additional information resources about the topic. Figure 2.27 shows that topic maps can basically be considered graphs in this simplified form.

The interesting point now, which make topic maps a breakthrough in the representation of knowledge for computers is the fact that associations between topics can in turn be *reified* with other topics. This implies that each association can be further specified by another topic map, since these so-called reification topics can have associations to other topics as well. Topics have types that are also represented as topics. This schema, while sounding very complex, makes it possible to basically store any knowledge in computer-readable form and enabling the computer to infer from it (Park, 2002) because every associative relationship between any concepts can be modeled as in a brain.

This allows the technology to serve as a useful tool when developing knowledge extraction/management systems.

2.7 Summary

This chapter gave an overview of the basics and technologies necessary to fully comprehend this thesis. I have introduced the basics to networks in biomedicine and presented advanced computer

technologies for big data processing (MapReduce and NoSQL). Also, the necessary natural language processing techniques required to build state-of-the art text mining systems (as described in the next chapters) were explained. Finally, I also gave an introduction to the topic maps concept and the representation of knowledge in subject-centric computing. When appropriate, I have given references to later sections of this thesis in order to provide a deeper understanding of why the respective aspect was introduced.

I hope, this chapter could show that to use the discussed technologies, a deep understanding of the mechanisms how computer systems such as Hadoop and HBase are implemented is necessary in order to be able to devise algorithms that can utilize them efficiently.

3 Text Mining in a Nutshell

In the preliminaries (Section 2.5), I have introduced various computational linguistics and natural language processing methods. These individual techniques can be applied to text data but still do not enable a computer to fully extract contained knowledge. In order to achieve this, the techniques have to be applied in a combined approach in order to structure unstructured text information.

Text mining (TM) can be defined as a technique from natural language processing to extract assertions about facts from unstructured text documents. The idea behind (usually) is that a computer is taught to read and understand texts written by humans and parse them into computer-readable format (see Figure 3.1).

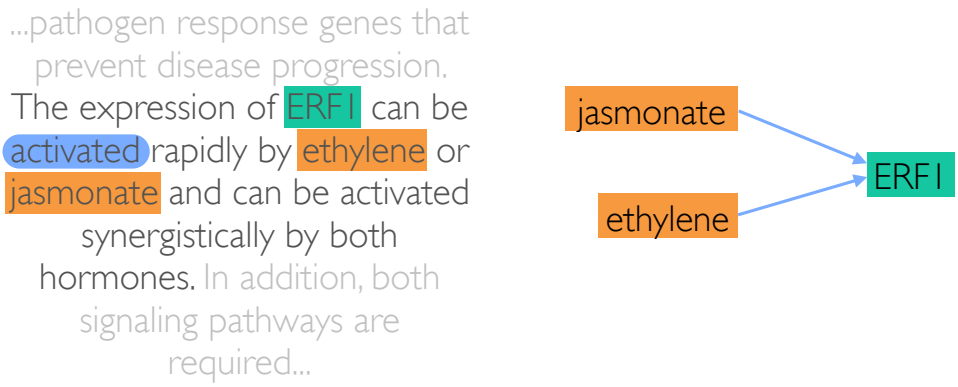


Figure 3.1: Text mining principles. Text is deconstructed in a way that (a) factual entities (ERF1, jasmonate, ethylene) are identified and (b) assertions between those entities (activates) can be extracted. Extracted assertions can then be displayed as nodes with links giving the type of interaction. Green: genes, orange: metabolites, blue: an activating relation. The example text was published in Lorenzo et al. (2003).

This is obviously a non-trivial task. This chapter will therefore give a brief introduction to *text mining* in general and text mining for biomedicine in particular, discuss different approaches and describe the theoretical foundations of implementing a full text mining system (as then executed in Chapter 4).

While I will use the terms biomedical TM and TM synonymously, the concepts introduced here are valid for both. The main difference between the biomedical and other domains is a shift in text data and ontologies, but the technologies remain identical.

The concepts introduced in this chapter are generally independent from any tangible implementation. The adaptation of the approaches to distributed computing will be covered in Chapter 4.

3.1 Text Mining Approaches

The various techniques for text mining can be broadly classified into four categories: *Co-occurrence* based methods that attempt to connect biomedical entities appearing together within the same fragment of text (e.g. a sentence), *semantic* or *rule-based* systems that try to extract relations between entities based on computational analysis of the written language, and *statistical* systems that try to find patterns in text frequency data Cohen and Hunter (2008). Last but not least there are *sentiment* analyzing techniques that measure the mood of statements and bring those into context. As in polyglot persistence (Section 2.4) the techniques can be combined in any way in order to build a more powerful text mining system.

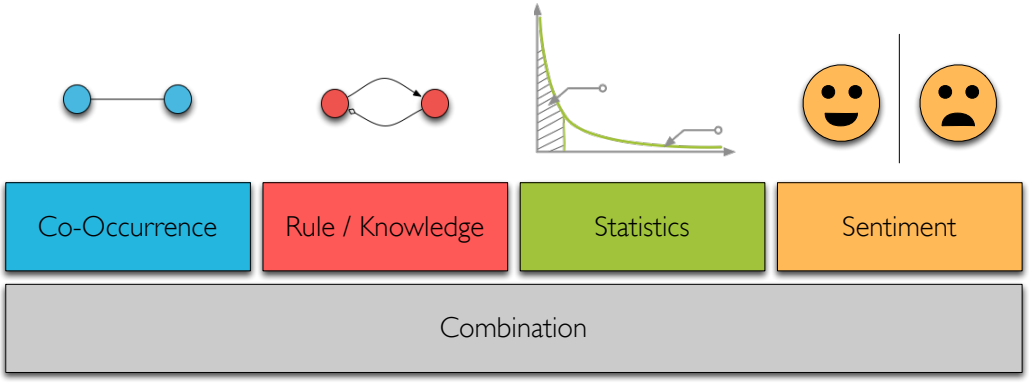


Figure 3.2: Four basic types of text mining. All types extract knowledge from text in one or the other form. A combination of any of the four approaches can be called polyglot text mining.

3.1.1 Co-Occurrence Based Text Mining

Co-occurrence based text mining methods basically look for single or multiple words that occur within the same unit of text (e.g. a sentence or a paragraph) and then posit an interaction between them. This interaction is not specified further and only denotes that the words of this tuple occur together within a specific set of boundaries. While some text mining experts might not consider co-occurrence based methods as true text mining, it has some advantages. Most importantly, it is a very simple approach that does not need any linguistic, syntactical, or semantic information other than to know what is a concept within the text and what not. With already that little information a lot can be asserted about the underlying text.

The main focus of work with co-occurrence based systems it to detect features in texts by term frequency analysis. If two terms interact with each other in a co-occurrence level, they are more likely to be of significance to each other. A lot of work has been done to develop scoring systems to quantify the significance of term co-occurrences (Wren, 2004) and also within the work of a master's student under my supervision (Gross, 2011).

On a database level, co-occurrences can be stored as matrices where each word is covered in each dimension of the matrix and the frequency of the combined occurrence of the tuple of both words

is indicated in the respective cell. This allows an easy integration of mathematical matrix methods to the frequency of interactions. Such matrices are symmetrical by definition, as a co-occurrence is an undirected interaction. By row or column operations, co-occurring terms can be used to promptly display so-called tag clouds (Kuo et al., 2007; Oesper et al., 2011) that generate a visual display of the interaction partners of a specific term. Usually neighboring terms are organized in an organic way with terms having more evidences for co-occurrence displayed in bigger font sizes. Various other applications are possible, for example, Cohen et al. (2005) used co-occurrence matrices to determine synonymous terms by analyzing the network structure the matrix imposes.

Several co-occurrence based text mining systems have been implemented. iHop[50] (Hoffmann and Valencia, 2004), for example, gives a user the possibility to search Pubmed[51] abstracts for genes and diseases and produce a list of co-occurring relationships.

Various approaches to bring co-occurrence functionality efficiently to MapReduce-based (see Section 2.3) applications have been developed. Fontoura et al. (2011), for example, developed an approach to efficiently store co-occurrence matrices in inverted indices and Lin (2008) implemented an efficient algorithm to compute co-occurrence matrices with MapReduce (see also the “pairs and stripes” approach, Section 2.3.1).

3.1.2 Semantic and Rule-Based Text Mining

Rule-based or semantic text mining methods attempt to use some sort of linguistic knowledge about how language is structured, (biological) facts are represented and what types of relationships are possible between entities. They incorporate semantic knowledge. While being more difficult to process than plain co-occurrence methods, the gains are usually worth the efforts. An exact specification of *how* different concepts/entities interact with each other is crucial in understanding what is really meant by the text. Semantic systems need additional information from text such as semantic role labeling data.

There are a multitude of systems possible that use any kind of prior knowledge in order to extract facts. Ranging from simple regular expressions determining specific assertions to sophisticated artificial intelligence tools that attempt to extract relations between entities and more. Some related work can be found in Section 3.2. The pipeline for a TM system that is introduced in 3.3 and executed in Chapter 4 also give more information on the topic.

3.1.3 Statistical Text Mining

Statistical methods can either work with classifiers on any level within a text and then output probabilities for specific interactions or use frequency data of chunks in texts. Typically they require large amounts of expensive, labelled training data in order to perform well. Methods solely based on frequencies of chunks (e.g. words) are usually less expensive but rarely can output relations between entities as co-occurrence and semantic methods do.

[50] http://www.ihop-net.org, accessed 2012
[51] http://www.ncbi.nlm.nih.gov/pubmed/, accessed 2012

These methods, however, can be used for a variety of applications in statistical text analysis. The analysis of word frequencies led to the discovery of so-called *stop words* (Section 2.5.1). An investigation, how the frequency of words behaves in the biomedical domain was conducted (Rebholz-Schuhmann et al., 2005).

Just to mention some examples for statistical TM systems, various articles were published in the early 1960s that conducted analyses on the original authorship of various letters and manuscripts (Brinegar, 1963; Mosteller and Wallace, 1963). The authors of the studies used various statistical measures, such as word length, sentence length, word frequencies in order to determine the original authorship.

These techniques from statistical text mining can also generally be used for categorization of text (e.g. Bayesian classifiers for spam detection (Sahami et al., 1998; Winkler, 2011)) or representation of document similarities through bag-of-words (Manning and Schuetze, 1999).

Other systems use statistical methods to implement a classificator for sentences. Agarwal and Yu (2009), for example, published a system that classifies sentences whether they belong to the introduction, method, results, or discussion categories of a research article.

According to Jarman (2011), statistical text mining is mainly about the identification of patterns in text and the inference of rules from that. Besides that, it can also be used to determine general inferences about language, as will be seen later (Section 5.4.1.)

3.1.4 Sentiment Analysis

In the last years, another technique for text mining has emerged. Instead of extracting facts from textual data, computers are taught to understand emotional feelings that were expressed by the author of a text through the choosing of his/her words (Wright, 2009). This so-called *sentiment analysis* provides a way to determine the mood of certain words and assign them negative, neutral or positive sentiment. For example negative words might include “terminate” or “deteriorate”, while positive words could be “achieve” or “efficient” (Loughran and McDonald, 2011).

In order to determine the kind of sentiment associated with different english words, Bradley and Lang (1999) have performed a study where subjects were presented with various words and their reaction to the three categories pleasure, arousal and dominance was measured. A score for each word was then given and with this information a combined measure for the sentiment of words could then be computed.

Applications with sentiment analysis have been mainly developed within the psychological domain (Pestian et al., 2012) and for detecting emerging trends, for example for pandemic analysis (Chew and Eysenbach, 2010). As another example for the non-scientific community, the sentiment of Twitter tweets has been used to predict trends in the stock market (Bollen et al., 2010).

Another application of sentiment analysis is the detection of subjectivity and objectivity in statements, which can be useful to discriminate written text according to facts or opinions of the author and therefore assess the certainty of specific statements. The higher the mood values in either direction (positive or negative), the more subjective the opinion is and by the same token, the more neutral the mood values are, the more objective is the statement (Liu, 2010).

While I will not go into detail here, the architecture of Excerbt (as described in Chapter 4) allows for an easy addition of sentiment analysis systems in order to attribute extracted relations with a mood-score. This has been done in separate projects that will not be described here. A more extensive introduction into the field of sentiment analysis is given by Liu (2010), Jaganadh (2012) and deHaaff (2010).

3.2 Related Work

The biomedical research community has a long track record of working in the field of text mining. In the following, I want to give a short history of text mining and introduce some popular systems, their advantages and disadvantages.

Most text mining systems incorporate basic co-occurrence information because of the simplicity of the approach. The need for semantic queries going beyond the possibilities of plain co-occurrence based or statistical methods was soon discovered and led to the development of semantic search engines like Powerset (Converse et al., 2008) or Swoogle (Ding et al., 2004). Relation-extracting systems that take into account, how language is structured, which entities in a text actually interact, and how those entities influence each other are, however, still in their infancies due to the immense complexity of the task despite the fact that they have been around for a couple of years now. Many existing systems present disadvantages regarding the types of entities extracted or lack sufficient tools in order to build qualitative models from search results in order to structure the complex knowledge that the user is desperately trying to find. More widely known systems, such as MEDIE (Miyao et al., 2006), iHop (Hoffmann and Valencia, 2004), or Chilibot (Chen and Sharp, 2004) only focus on specific entity types, such as protein-protein or gene-disease interactions or make generating network views of retrieved data in order to build qualitative models a tedious task. Another system PASBio (Wattarujeekrit et al., 2004) uses semantic role labeling data and extracts relations based on 30 words.

Table 3.1 gives a comparison of some select text mining systems according to the criteria of the types of entities, the types of interactions between such entities, the kind of texts analyzed and the speed of the user interface. It becomes evident that most systems restrict their types to a specific set and do not provide a good user experience. Also, most systems do not include all possible freely-available text resources for biomedical articles.

A recent publication by Ananiadou et al. (2010) demonstrated that the extraction of relations solely on the basis of relations between entities through verbs (e.g. gene-A activates gene-B) leaves a large number of interactions unidentified. The authors therefore suggest the more general approach of extracting so-called events that additionally take non-verb interactions between entities into account. For example, in the sentence “Gene-A, an upstream regulator of gene-B, causes disease-C.”, the relation-based knowledge extraction approach would correctly identify the relation “gene-A causes disease-C” but it would miss the additional event “gene-A regulates gene-B” that is implicitly contained in the subordinate clause of the sentence. An event-based extraction system would provide a generalization of the concept through the localization of assertions (i.e. relations and events) about facts (entities) linked through any semantic relationship, not strictly those based on verbs.

	iHop	Chilibot	EBIMed
Entity Types	genes, MeSH-terms, chemical compounds	genes, keywords	genes, GO annotations, drugs, species
Relation Types	co-occurrence	interactive, parallel	co-occurrence
Analyzed Texts	all PubMed	30 newest PubMed abstracts	500 newest PubMed abstracts
Speed	fast	slow	slow

	MEDIE	BioContext
Entity Type	genes, diseases	genes, anatomy, species
Relation Types	syntactic relations	gene interactions, localization
Analyzed Texts	all PubMed	all Pubmed, PMC
Speed	medium	medium

Table 3.1: A comparison of select text mining systems. TM systems have been compared by the type of entities they support, the type of relations that can be detected between entities, the text sources and the speed of a query against the system. Based on Barnickel (2009) and own evaluations.

Some systems already have been implemented that extract events, but on a limited scope. Miwa et al. (2010) implemented a machine learning approach that extracts binding and regulation events in biomedical publication. It has been improved recently by additionally implementing coreference resolution and domain adaptation approaches (Miwa et al., 2012).

3.3 How to Develop a Text Mining System?

Developing and implementing a full-fledged text mining system is a non-trivial task. Apart from having to obtain specific textual resources (which will not be covered here) and deciding which combination of approaches to implement, at least the following steps have to be dealt with. This is a generalization of the concepts described in Barnickel (2009).

1. Documents have to be parsed. Any text is usually given in some kind of computer-readable format and has to be brought into another format ready for further processing. This step also includes preprocessing steps, such as extracting all valuable information that is stored in a structured way and normalizing different character encodings and so on.

2. After documents have been processed, contained texts have to be extracted and split into single sentences. Since sentence splitting is not a straight-forward approach, a heuristic method that was trained on similar text data has to be used.

3. Once we have single sentences, we start the process called Named Entity Recognition (NER). This searches for named concepts and extracts their positions within the sentences. Examples for named entities include gene names, disease names, tissue names etc.

4. The next step is then to determine the part-of-speech tags for each word. This means, we determine for each word whether it is a noun, a verb, an adjective, a determiner, etc. Afterwards this information can be used in order to identify semantic roles that give the meaning of a sentence.

5. With this information we can now determine who in a sentence did what to whom and also extract additional properties such as where or when and store this information in a database as nodes and edges. A simple approach that only takes part-of-speech information into account could be to simply take the verbs in a sentence and take a look at identified concepts left and right of the verb.

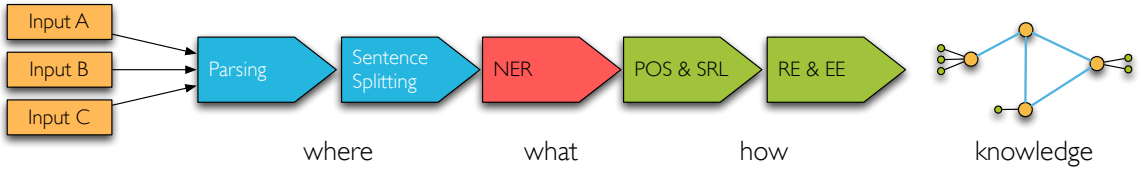


Figure 3.3: Core steps in text mining. First, resources worth processing have to be found (input). Through the various processing steps, assertions such as where in a text can what how interact. The output is then in a form of knowledge graph (in our case a topic map) structure.

In Figure 3.3, a simplified summary of these steps is displayed. Basically, first we need to get information (where), then we need to extract relevant concepts that we're interested in (what) and finally we need to extract relations and events between those concepts (how). Steps 1 and 2 correspond to information retrieval, step 3 is the named entity recognition part, and steps 4 and 5 correspond to the relation and event extraction.

The necessary single step techniques have been introduced in Section 2.5. Fundamentally, text mining can be seen as a combined application of various techniques from natural language processing. While often being mentioned without the reference to NLP, the purpose is to build a system that is capable of extracting knowledge from text and store it in some kind of database in a way that a user can efficiently access the information and deductively reason with that information.

The complete process is illustrated with an example in Figure 3.4(a). In this example, a hypothetical system identified that "phosphorylate" is a verb, that LRRK2 is the causer (or agent) of the action and that 4E-BP is the receiver (or patient) of the action. Additional information can be attributed to the statement "in vitro" which gives us a location where the action takes place. The modal verb "can" can also be used to further specify the relation the two entities stand to with each other. It tells the reader that the stated assertion is of hypothetical nature and that the author is not absolutely sure that this interaction really takes place, only that it can take place. The information stored as a graph would then resemble Figure 3.4(b).

LRRK2 and 4E-BP are the nodes, "phosphorylates" is an edge with two additional properties as a context to the edge, a location (in-vitro) and that the statement is hypothetical (can).

The Excerbt system, as extensively described in Chapter 4, basically implements exactly this process and follows these implementation steps in order to convert unstructured text into a structured format based on the topic maps and associative thinking paradigms. The system as developed here sets, however, a main focus on scalability to exponentially increasing data sets, which makes it unique in that regard.

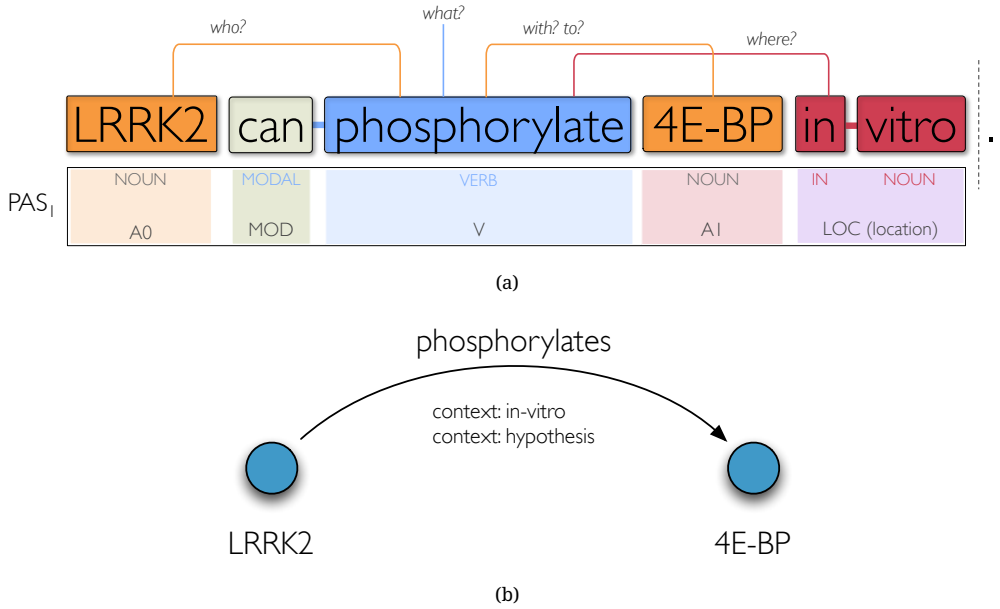


Figure 3.4: Text mining on a sentence level. (a) Steps performed on a single sentence include part-of-speech tagging, Semantic Role Labeling, and Relation/Event extraction. These steps are necessary in order to compute *who* is doing *what with/to* whom and *where*. Modal arguments (such as "can") can be used to determine the weight of the interaction: How sure was the author that the relation is true? (b) A sample edge extracted from the above sentence as stored on a computer.

3.4 Obstacles

As with all computer systems that try to handle texts written by humans in natural language[52], the task of extracting relevant information is obstructed by various obstacles that will be briefly mentioned in the following.

Starting from relatively straight-forward problems, texts have to be split up into single sentences in order to extract the basic structure of the articles. The usual convention is that a new sentence begins with a capital letter and the sentence before ended with a punctuation character, such as period, exclamation mark or question mark. Between the punctuation character and the next letter usually is a whitespace character. Some authors (due to typing errors) omit that whitespace, making a naive approach that parses for occurrences of the pattern very difficult. Added to that, the convention to begin a new sentence with a capital letter is ignored in some communities where, for example, specific gene names have a small letter at the beginning and are written as such in the text even if they are the first word of a new sentence. The existence of abbreviations, such as "etc." or "i.e." also contributes to this problem. A vanilla approach that takes these additional considerations into account with additional manual rules can detect about 95% of the sentences correct (O'Neil, 2008).

The solutions that are favored today and provide a better performance often revolve around a heuristic approach trained on sample training sentences and contain different language models. An example for

[52] in contrast to artificial languages, such as programming languages

such a method would be the maximum-entropy approach by Reynar and Ratnaparkhi (1997) or the one that is used by the LingPipe NLP library for Java[53].

One of the most important problems is the *ambiguity problem*: there exist multiple relationships between the actual language and the meaning of a word. Ambiguities exist at every level of linguistic structure. For example: in an arbitrary sentence that mentions the word "fat": is the fat a noun or an adjective? If it is a noun, which concept does it represent, that of the tissue or is it a gene name?

The ways to express a certain concept are countless: One could use alternate symbols for the same concept, such as flying saucer or UFO (or in biology: BRCA1 or IRIS), there are different spelling variants of the same symbol, such as BRCA1, BRCA-1 and BRCA 1 that can not always be directly mapped to the same concept.

Also, consider the following two sentences:

1. She boarded the plane with two suitcases.
2. She boarded the plane with two engines.

Those two sentences look exactly the same on a syntax level. For a human it is inherently evident that in the first example the two suitcases belong to the woman, whereas in the second example they belong to the airplane. For automated knowledge extraction methods, this poses a big problem: A computer does not (yet) have the background knowledge that a woman cannot have engines. The inherent meaning of the words and their unstated relations to each other are unknown to a computer system.

Another problem, especially for the biomedical domain is the determination of the species of a gene name. In other words this is the attachment of a context to a gene. There exist several approaches how to deal with that kind of contextual information, such as Harmston et al. (2011).

Abbreviations in text also poses a significant scope for error, not only for sentence boundary detection. Usually, abbreviations are introduced in the beginning of a document with a structure something like "This Is An Example Abbreviation (TIAEA)". Detection of such abbreviations through detection of mentions of the short variant in brackets and the long variant before that with multiple capital letters, works obviously in many cases, but not all. Multiple studies have been conducted on the detection of abbreviations (Okazaki and Ananiadou, 2006; Okazaki et al., 2010; Schwartz and Hearst, 2003), because replacing the abbreviation in subsequent sentences with the full variant is a useful technique for text mining.

While to some of those obstacles, such as the ambiguity problem, there were no solutions available at the time of writing this thesis, others, such as alternate symbols are relatively easy to implement. If we extract relations as described in this chapter from unstructured text, context attributes can be extracted from the text as well and be used in order to counteract some of these obstacles.

[53] http://alias-i.com/lingpipe/demos/tutorial/sentences/read-me.html, accessed 2012-07-02

3.5 Summary

In summary, text mining is the application of NLP technologies in a combined system in order to extract knowledge from unstructured texts into a form that is later processible by computers. The general pipeline, how to implement a text mining system is clear, but the devil is in the details, as always.

In the next chapter, we will use the NLP techniques from the preliminaries and the TM approach as discussed here in order to implement the text mining system Excerbt on a large scale for massive volume data sets.

4 Excerbt: Next Generation Biomedical Text Mining with Big Data Approaches

> "We will not be limited by the information we have.
> We will be limited by our ability to process that
> information."
>
> *Peter Drucker*

In 2009, when Thorsten Barnickel finished his PhD thesis (Barnickel, 2009), a new text mining system, called EXCERBT was introduced, that differed from then existing, more traditional systems in some points. Contrary to other systems on the market it had the following characteristics: On the one hand it supported a large number of different biomedical entity types. Most systems at that time supported only genes and maybe phenotypes (Section 3.2). With this system it was now possible to also include other types, such as microRNAs, SNPs, metabolites, etc. On the other hand it supported a large number of different relation types. Again, most systems at that time were limited and most often only considered co-occurrence, but no relations that were based on the semantics of texts.

While the first point, the support of multiple entity types, is merely a problem of choosing and implementing the right ontologies when using a dictionary based NER approach, the latter is more difficult to accomplish. Generally, defining the type of relation between any biomedical entities within a text, requires a processing software capable of understanding the semantic structure and meaning of texts. In Excerbt, this was achieved through the semantic role labeling engine SENNA. All Pubmed abstracts and freely-available articles in Pubmed Central have been processed with the tool that was designed to be fast. Only then could the expensive step of SRL be performed in acceptable time. After that, relations based on verbs were extracted for entities in the `ARG0` and `ARG1` roles of extracted predicate-argument structures.

The Excerbt text mining system is the basis of my work within the scope of this thesis. In the beginning of this chapter, I will therefore investigate its shortcomings and provide solutions that are later combined into a fully functioning system.

Consider, for example, the daily increase in citation volume in PubMed MEDLINE (see Figure 1.1) at the alarming rate of nearly three new publications per minute that we have since the beginning of 2012. The original Excerbt was not equipped to handle these data increases and therefore a completely new architecture had to be invented. This realization came directly according to the quote by Peter Drucker: "We will be limited by our ability to process that information.".

The original system was based on a single relational database on a single server and associated problems (as outlined in Section 2.4) soon became evident. With the increasing data volume, an update of the data basis became consistently more difficult each day. Hard drives filled up, the speed at which the

system could be updated declined massively with every new piece of data and also read operations for user queries became slower and slower.

For these obvious and other less obvious reasons as described in the next section, I developed a completely new approach for handling text mining data on a massive scale. The core idea is to use an approach that considered parallel operations and the *distribution of data* to a cluster from the ground up combined with a maximum of flexibility in all involved steps. Recently released software systems that became widely used throughout the industry since then, such as Hadoop and HBase, combined with the processing framework MapReduce set the basic foundations of the new system. Every component of the new approach is scalable and can be easily exchanged for other systems and tuned individually. This not only improved the processing workflow as described by Barnickel (2009), but also allowed the addition of various other approaches extending upon existing ideas and implementing completely new subsystems for gaining a better understanding of the massive amounts of underlying data.

Another major requirement for the new system was an effortless incremental update procedure. Due to the steady increase in data volume every day, an update every few months was not feasible and instead hindered any work done with the system by biologists. I also wanted to retain the policy of the first version for supporting large numbers of different entity and relation types and therefore put an effort into making the system completely independent from any domain knowledge. This then allowed the peaks into other domains such as economic news that have been experimented on in separated projects, but whose description would go beyond the scope of this thesis.

The underlying problem here was that the first Excerbt's approach was completely document-centered in the implementation. This means that a single document could be seen as one operation throughout the system. On the other hand, if a single new entity was added to the system, we had to perform again a document-centered named entity recognition, i.e. searching all documents whether they contain the new term instead of searching the term in the documents.

In the following sections, I will first roughly describe the old system and uncover several problematic parts that needed improvement. Then I will outline the new approach and describe specific subsystems in detail that changed significantly from before, mainly the named entity recognition (NER) and the semantic role labeling (SRL) modules. I will also go into details about the impacts that MapReduce and HBase had on the system. Finally, I will briefly present the system from a user perspective.

The basic workings of a text mining system such as Excerbt have already been outlined in the previous chapter. In the following will concentrate on specific areas of the program that need further explanation or deviate markedly from the presented implementation path from Section 3.3.

4.1 The Original Excerbt

The original Excerbt system lacked some properties to be of true use in our current big data economy. Figure 4.1 gives a broad overview of the workflow that was necessary in order to keep the system up-to-date. Most obvious is the need to manually manage three distinct systems (the PostgreSQL[54] database server, the cluster responsible for sentence splitting and the cluster for indexing and named

[54] An open-source, enterprise ready relational SQL database, http://www.postgresql.org/, access 2012-08.

entity recognition) and keep them manually in a consistent state with each other. In order to update the system with new text data, the following steps also had to be performed by hand.

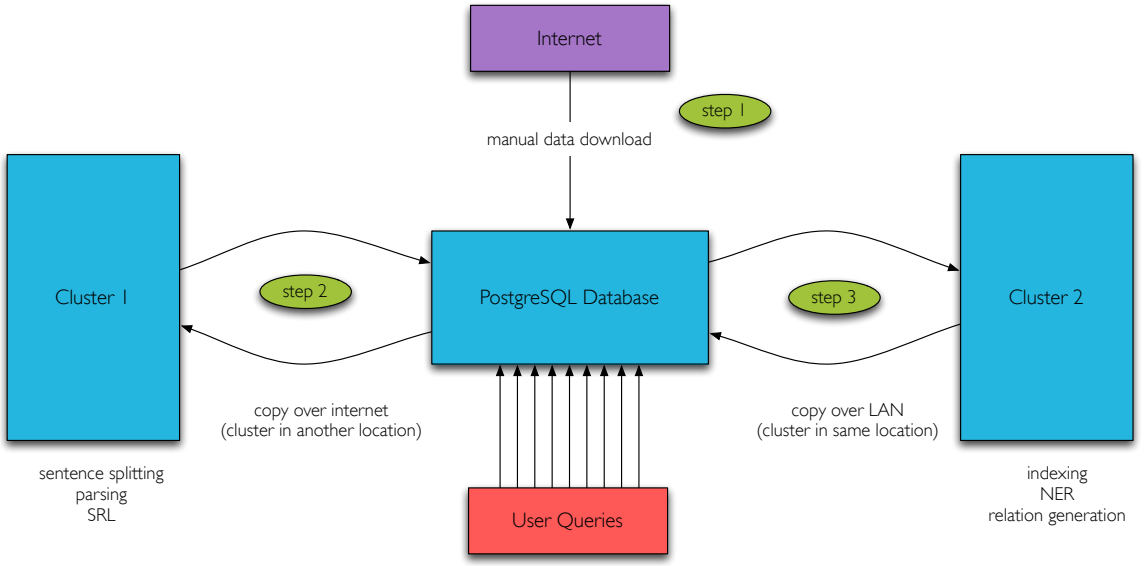


Figure 4.1: Workflow of the original Excerbt. Hardware elements are colored blue, software elements green.

1. Download new text resources from the resource's web site. In this case, this implied a download of all archive files from the Pubmed and Pubmed Central FTP servers.
2. Extract the data onto the database server.
3. Copy the data and program code to Cluster 1 and perform each processing step manually. Check for completeness and consistency after each step.
4. Copy processed data back to the database server.
5. Copy new data and program code to Cluster 2 and perform indexing and named entity recognition steps manually. Again check for completeness and consistency.
6. Copy processed data back to the database server.
7. Make sure the user interface accesses the right tables on the database and correctly works with the newest software.

These steps illustrate that the amount of manual work was tremendous in order to update or setup the system, especially tedious copying of files around different cluster systems. The time needed in order to update the system amounted to a considerable sum, in the span of a few weeks (about three to four) for all involved steps. The reason that there were two different cluster systems instead of only one performing all of the tasks was that Cluster 2 was not powerful enough for all the involved SRL steps. The two clusters were physically separated and could only be reached via Internet, which means that a use of the more powerful, external cluster, was out of the question for all processes.

The major point, however, was the fact that the system was built upon a single database server that held all data from input to relations output. Due to this architecture, standard modeling strategies for relational databases were removed soon due to performance reasons. Database schemata were denormalized and redundantly stored on the system in order to limit the number of expensive join operations and to enable a better performance by sharding individual tables onto different hard disks of the server. Joins could, however, not be fully removed for various queries. When viewing this in a bigger context, the basic design principle behind relational databases, non-redundant storing and normalized schemata, were violated, and as such the restriction to use a relational database model in the first place became obsolete. Even with these relaxing constraints, the architecture still proved to be too strict for our use cases in an environment where changes to input data and tasks were the norm and not the exception. This obviously hindered any natural evolution of data and algorithms.

The original database schema had relational tables for each type of data, such as documents, sentences, predicate-argument structures, hits, relations, etc. After parsing sentences and storing them in separate tables, these sentences were role labeled. Extracted PAS were stored in turn in a separate table. Tables for verbs, entities, entity types, relation types and their connections to each other were also stored in a relational way. All relations between different tables were modeled according to the traditional relational approach with either direct ID to ID mapping for a 1:1 relation case or with separate mapping tables for the 1:n, n:1 and $n : m$ cases.

These points clearly describe that scaling the data volume by use of the existing approach through scaling of the data model and the architecture is a difficult to impossible approach because it was optimized to operate on a single server. Such a server can be outfitted with better and more specialized hardware. This, however, does not solve the problem but merely delays the point where other measures need to be taken.

From a user perspective, the system also reached its limits. The users' increased demands to Excerbt simply could not be fulfilled by a single server in the long term as hardware restrictions limit the amount of user interaction that can be performed at any given point in time. Ad-hoc queries (any query that is not pre-computed by the workflow) could not be performed at all since such queries had to be formulated as long-running SQL join operations that completely blocked other operations on the server.

4.2 A Novel Approach

All these reasons made it obvious that the approach had to be radically changed in order to be able to cope with the increasing requirements of a state-of-the-art text mining system. First of all, every component had to be put into one integrated system. Three different handling systems are hard to manage code-wise and for administrative reasons. Further, the limitation to a single database server had to be lifted. Increased demands on the system on all levels (support for more concurrent users, more concurrent request), increased ad-hoc query complexity and ultimately the tremendously increasing data size all made the requirement for solutions that could handle the expected loads and were at the pulse of time obvious.

A completely new and different approach was needed for the underlying technology. Excerbt had to be re-built from scratch. Focus had to be laid on scalability of the architecture. SQL for the access to data is a long-proven standard when implementing database systems. Its easy of use and the familiarity of most developers make it in theory an ideal candidate. Because of this, the new system would have to implement a relatively easy programming structure to ensure a future maintainability, if compliance with SQL has to be dropped. Cluster architectures usually can become complicated in terms of communication between the nodes. Message-passing techniques that are commonly used in such environments are complicated from a developer perspective. We needed built-in possibilities to deploy the processing workflows onto a cluster and finally, the hardware should provide low acquisition and upgrade costs, meaning the employment of standard hardware instead of highly specialized, expensive equipment (see also Section 2.2, Figure 2.4), which basically fits the properties of a cloud computing based system.

At the beginning of my doctoral thesis project, alternative data processing systems such as Hadoop and NoSQL databases just made their first appearances in professional enterprise environments and became more prevalent in the IT community as they were just before their initial stable releases. After careful evaluation of multiple technologies, Hadoop proved as a viable candidate to replace the data management system and offered all the possibilities that were needed in order to improve the system. It put an end to the increasing storage and processing requirements because it was created for commodity hardware and promised simple extension. The underlying distributed file system allowed us to store the data in a format that significantly improved data throughput by knowing which servers in the cluster have what data and therefor be able to optimize computations to local machines. Especially in combination with MapReduce, all of the problems we had with the original system vanished because we now had a single cluster that did all of processing and all of the data storage. Also, because of the adoption of the Hadoop framework stack by famous companies, such as Facebook or Twitter, it suggested a promising future. Compared to other, similar projects such as Cassandra[55] or Hypertable[56] that came up around the same time and could have served as valid alternatives, Hadoop and HBase already had a much bigger community, making it the obvious and preferred choice.

The alternative of retaining the PostgreSQL stack was rejected, because ultimately, the automatization of all involved manual steps proved to be impossible to implement. In theory, it could have worked to extend PostgreSQL in order to support replication and sharding among different servers. However, this would have clashed with some of the earlier arguments: a simple programming model and the future maintainability. Replicating and sharding of a PostgreSQL database is still a tedious task (Indelicato, 2008) and therefore it also was no viable option. Also, an extension to a sharded relational database would have improved only one side of the coin: the data storage. Data processing still would have to be outsourced to a separate cluster, leaving the maintenance costs of semi-automatic data handling on two different systems at a high level (see Figure 4.1).

In defense of the implementation of the first version of Excerbt, it is to say that advanced technologies such as Hadoop or HBase were not accessible at the time then and the primary focus of the system was on proof-of-concept. Creating a prototype that proved that the idea with semantic role labeling worked and that it is possible to extract knowledge with this approach. The focus of my work is, however, in the general applicability and scalability of the system and a subsequent enabling of other, more

[55] http://cassandra.apache.org/, accessed 2011
[56] http://hypertable.org/, accessed 2011

sophisticated data analysis strategies. The main problem of big data needed to be tackled and the system had to be rebuilt in a way that ad-hoc queries of any kind and an easy extension of any approach could be done without major problems.

The new system therefore basically revolves around two central components: A single cluster outfitted with the NoSQL database HBase for storage of all data (raw, intermediate, and processed) and Hadoop MapReduce for data processing. For each step in the workflow of the text mining system as outlined in Section 3.3, from resource parsing to graph generation through the relation extraction engine, MapReduce jobs were implemented. All intermediate results are stored in HBase in separate tables. User access is handled directly by a wrapper to the HBase database, ensuring a scalability not only for data, but also for the number of users. The table schema has been retained in the general concepts on what to put into different tables, but has been completely rewritten from scratch internally in order to make use of HBase and MapReduce efficiently and provide fast access to any required data (see Section 4.5).

Architecturally, Excerbt is implemented as a multi-tier architecture and written in the Java programming language. An overview diagram is depicted in Figure 4.2. Excerbt consists of six layers that basically represent the modules data storage, data access, management layer, data retrieval, data generation, and data presentation as described in the following.

Data Storage

The data storage layer is responsible for storing and actual batch-processing of data. It provides the interface between hard- and software through the Hadoop and HBase framework stack. Hadoop's and HBase's master and slave server processes run on actual hardware in this layer (NameNode, DataNodes, JobTracker, TaskTrackers, ZooKeepers, HBase Master, HBase RegionServers). See Section 4.2.1 for details on the used hardware. Jobs for de-novo setup and updates are submitted here, ad-hoc analyses can be done directly with Hive.

Data Accession

The data access layer provides data access objects and data-access-entities, common design patterns used in object-oriented Java Enterprise programming (Alur et al., 2001) for encapsulating the need to call data storage methods and functions directly. It also provides custom filters specifically implemented for working with Excerbt. See Section 4.5 for some particular details. Ad-hoc processing scripts and specifically needed hard-coded data are also within this layer.

Management

I have additionally implemented modules for managing the system. Specifically, I have developed a custom shell for managing the cluster processes and initiating update processes based on the HBase internal shell. The HBase shell[57] provides access to the data in HBase through a command line utility

[57] http://wiki.apache.org/hadoop/Hbase/Shell, accessed 2012-09

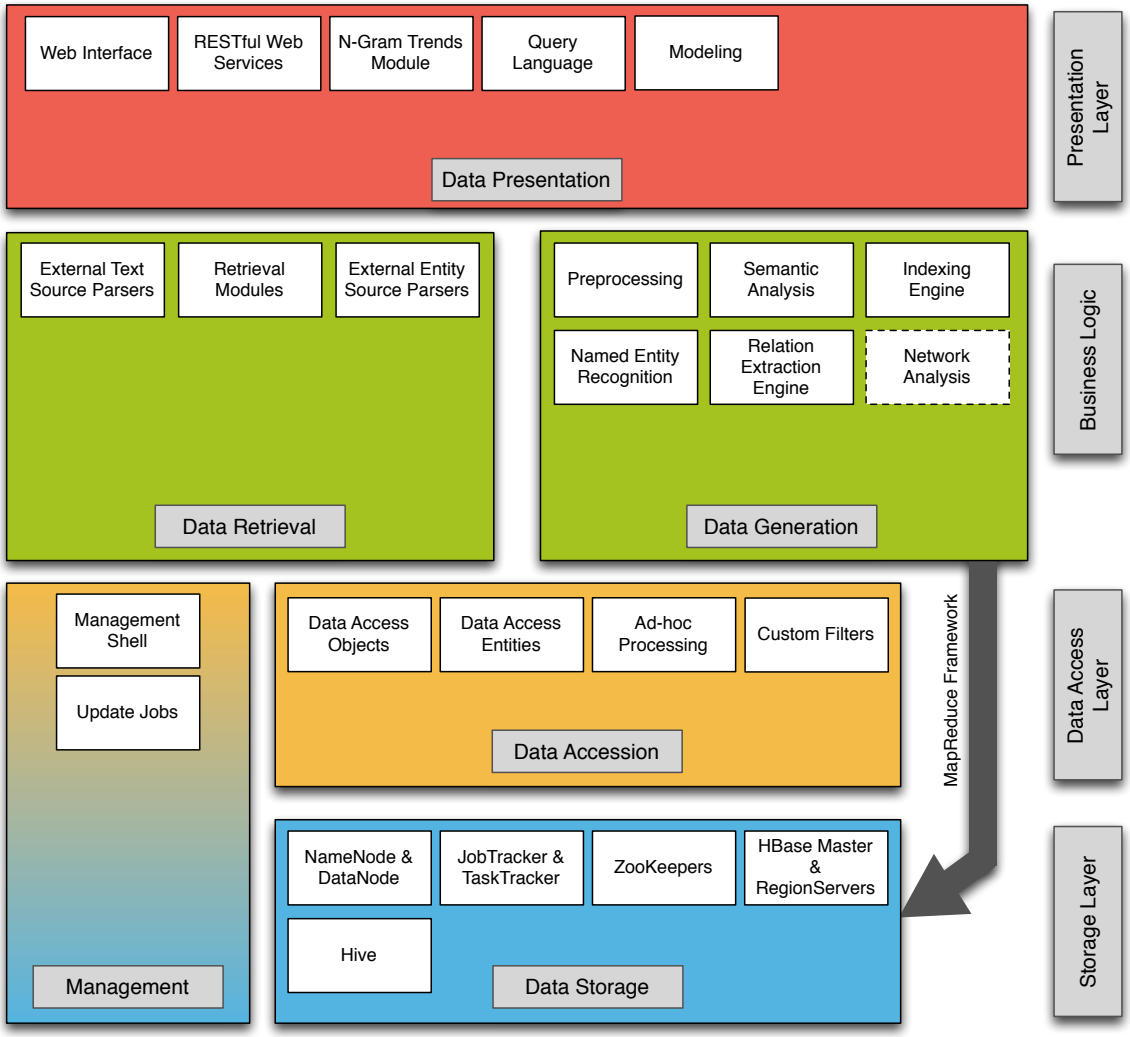


Figure 4.2: Excerbt architecture diagram. Excerbt is implemented as a multi-tier architecture. Blue is the data storage and processing layer (Hadoop and HBase stack), orange is the data access layer which provides interfaces to access data in HBase, green are two components for data retrieval from external sources and data generation via MapReduce. The presentation layer (red) is on a different hardware level and basically provides the RESTful WebServices and Web Interface.

with a special syntax, offering Scans and Gets to query the data. It is of great use for quickly checking the contents of specific rows and values, but cannot be seen as a real terminal to HBase as SQL-based systems offer. The Excerbt shell can be used by admin users to query raw data and execute custom methods to interact with the system.

Data Retrieval

The data retrieval layer contains the various subsystems responsible for retrieving and parsing external data sources. For each text sources (Pubmed, PubmedCentral, and OMIM) and for each ontology this implements the necessary parsers. A full list of supported ontologies can be found in Table 4.2 (page 78). The parsers all implement specific interfaces, so a later expansion for more ontologies or more text sources should not pose a problem.

Data Generation

The data generation layer is the main point of the system host the code for batch processing of raw data into relations in MapReduce. After preprocessing (parsing, sentence splitting, etc.), data is semantically analyzed via SRL. Indexing and NER modules as well as an engine to extract knowledge (relations) from processed data are also contained in this layer. See Sections 4.3 (Indexing and NER), 4.4 (SRL and Relation Extraction) for details. Multiple other jobs for ad-hoc analyses, N-Gram generation, etc. have also been implemented in this layer. Code and algorithms for network analysis, as described in Chapter 6 also belong to this layer. Data generation is on a parallel layer with data retrieval within the business logic as they both provide different methods to work with the actual data but do not provide data or user access themselves.

Data Presentation

The top layer finally provides access to the literature graph as processed by Excerbt. It contains the web interface, various RESTful Web Services and other modules for statistical trends and modeling. See Section 4.7 for details.

4.2.1 Hardware

The optimal hardware for Hadoop-based cluster systems is a science in itself. Hadoop was designed to work with so-called commodity hardware that has little to no protection against failure and which is commonly used in desktop and workstation environments. In contrast, such hardware is seldomly found among enterprise server architectures because of the aforementioned missing failure protection. Additionally, such hardware can usually be acquired at much lower costs than specialized enterprise hardware, for example for high-end database servers. The exact specifications for Hadoop-based systems, however, can be quite complex. The different server types in Hadoop and HBase have different requirements for the hardware. While master processes such as NameNode or JobTracker

usually do not need much hard disk space but have high requirements for memory. Slave machines that typically have the three client servers running (DataNode, TaskTracker, and RegionServer, Section 2.3.5) have a bottleneck in the amount of hard drives (the number of individual disks) and decent amounts of memory for processing, depending on the application.

For Excerbt, we chose to build a cluster with the following specifications. We opted for Dell PowerEdge R510 servers and bought identical hardware for all server types for simplicity. Each node has two quad-core Intel Xeon processors with 24 gigabytes RAM. They have eight hard drives with one terabyte capacity each, making eight concurrent I/O accesses to the data storage possible, correlating to the eight CPU cores.

At the time of writing this thesis (mid 2012) we have 20 servers in the cluster. The nodes are interconnected by a single Gigabit switch with two network cables for each computer in bonded mode, such that we have a total connectivity of two Gbit/s for each node instead of only one. The different master and slave server processes necessary for Hadoop and HBase to work have been distributed onto the machines as noted in Figure 4.3. The full cluster therefore has 480 gigabytes of memory, 160 terabytes of hard disk capacity, and 160 processor cores. These hardware properties have been chosen after considering recommendations by Hadoop users, such as Loddengaard (2010), and the tutorials on the Hadoop web site[58] as well as own evaluations with toy system configurations.

Figure 4.3: Excerbt hardware architecture. The current Excerbt cluster consists of 20 machines with two quad-core processors, 24 GB RAM, 8 $\times$ 1 TB HDD each. NN: NameNode, SN: secondary NameNode, HM: HBase Master, JT: JobTracker, TT: TaskTracker, DN: DataNode, RS: RegionServer, ZK: ZooKeeper. The machines are all Dell PowerEdge R510 servers. Total acquisition cost of the server was approximately 50.000 € in 2009/2010, which provided a very good cost-benefit ratio.

[58] http://hadoop.apache.org, accessed 2012-07-02

The reason for this particular hardware configuration lies in the determination of the applications the hardware should run later. There is a delicate boundary between the number of CPUs and the number of hard disks (I/O) that cannot be increased unlimited for both. A trade-off between the two had to be made. For Excerbt, we decided for a slight shift into the direction of better I/O capabilities (see Figure 4.4) in favor of jobs that needed to process more data but where the individual computations do not allocate much CPU time.

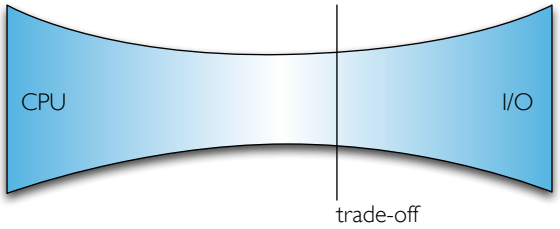


Figure 4.4: Tradeoff in Excerbt between CPU and I/O performance. The settling point of a cluster in the spectrum between a focus on CPU or I/O performance influences efficiency of running applications. For Excerbt, we decided that a balanced cluster with slight shift to I/O side would fit best for our use cases.

For smooth network operations, the cluster has been put behind a virtual network (VLAN). This means that every network operation is secluded from the general network since the cluster produces many requests through the network that can negatively impact regular machines but are necessary for the cluster to work.

Just as a comparison note: The old Excerbt system ran on a single machine with two dual-core AMD Opteron processors. The machine had 32 GB RAM and eight hard drives with 147 GB storage capacity each. The main reason for the low capacity was that the hard drives were fast, ultra-wide SCSI[59] hard drives. These are very expensive disks (compared to regular commodity disks, such as SATA) and only available in limited storage capacities because of an exponential proportion of storage vs. price. This old server is now in use for the presentation layer application processes.

This setup now gives the possibilities to develop a scalable text mining system based on big data principles.

4.3 Redesigned Indexing and Named Entity Recognition

Excerbt employs a dictionary-based approach to named entity recognition. The dictionary is constructed as a meta-ontology of the various resources as mentioned in Table 4.2. All terms are mapped to at least one entity type category, as listed in Table 4.1. While most of the resources annotated terms with their own categories, these have been manually categorized to fit Excerbt's entity types system.

A very naïve approach to a dictionary-based NER is to simply parse every sentence in a database for exact matching of any of the entities, for example via regular expressions or exact search through algorithms such as Boyer-Moore (Boyer and Moore, 1977). This, however, has several disadvantages. For one, the method cannot distinguish between various flexed forms of the words, such as genitives

[59] http://en.wikipedia.org/wiki/SCSI, accessed 2012

Entity Type	Description
Gene	Genes and protein names like BRCA1 or LRRK2 or p53. Note that those are strictly case-sensitive
Phenotype	mostly diseases, for example Alzheimer's disease
Cell Component	cellular components, like ribosomes, vacuoles, ...
Tissue	tissue types, e.g. liver, brain, but also cell types, e.g. HEK
miRNA	micro RNAs
Species	species names, such as Homo sapiens
Pathway	metabolic pathways
Metabolite	drugs and substances, such as Aspirin or helium
Domain	protein domain names
Environmental Factor	environmental factors, e.g. smoking, heavy metals
Person	mostly occupations
Individual	individually mentioned people
Method	biochemical methods, e.g. PCR
Mutation	DNA mutations, e.g. R1225A
SNP	single nucleotide polymorphisms
Function	protein functions
Phenomena	phenomena, such as death, mutation, or pregnancy
Geolocation	places
Object	solid objects, e.g. needle
Observable	observable facts, e.g. activities, functions, ...

Table 4.1: Ontology entity types. A listing and explanation of the various entity types in our meta-ontology.

Ontology	Extracted Entity Types
Entrez Gene	genes
KEGG Compounds	metabolites
OMIM	genes, mirnas, phenotypes
Brenda Tissue	tissues
KEGG Pathway	pathways
InterPro	genes
MPO	phenotypes
Swissprot	genes
Brenda Enzyme	genes
MeSH	phenotypes, metabolites, envirfacts, geolocations, methods, persons, phenomena, tissues
Mirbase	mirnas
NCBI Taxonomy	species
NIH environmental factors	envirfacts
KEGG Drug	metabolites
Lipidmaps	metabolites
Metabocards	metabolites
PubChem	metabolites
NCBI SNPs	snps
SnoMed	phenotypes, cell components, envirfacts, functions, geolocations, methods, objects, observables, persons, phenomena, species, tissues
Drugbank	metabolites
GO	functions
ICD10	phenotypes

Table 4.2: Ontologies included in our meta-ontology. Most ontologies have assigned only one entity type. OMIM, MeSH and SnoMed contain multiple different entity types and have therefore multiple types assigned. An explanation of the different entity types can be found in Table 4.1

and plurals. Other than that, this would be a highly inefficient form of searching. Better approach have been implemented by using indexing.

An index is a data structure invented for fast information retrieval. There exist multiple solutions for indexing without having to reinvent the wheel, the most famous being Apache Lucene[60] for Java, which has already been introduced in Section 2.4.2 or technologies revolving around inverted indexing. The differences will become clear later in this section.

Indexing and NER have previously been performed upon sentences and predicate-argument structures. The implementation was with plain Lucene and a build-up of multiple indices with an approximate same number of sentences or PAS, in order to enable a limited amount of sharding to the servers in a cluster for increased performance. For the new version, this process has been switched to the ElasticSearch (ES) solution (see Section 2.4.2). ElasticSearch helped to minimize Lucene configuration overhead and providing a useful solution to distribute indexing and search onto a cluster, which is handled completely independently by ES. The manual sharding steps could be omitted and the overall amount of manual maintenance dropped significantly.

Previously there were only two types of indices based on preprocessing of the text: a case insensitive and stemmed variant, and a case-sensitive and verbatim variant. *Case sensitivity* is the distinction between capital and small letters. *Verbatim* means that the named entity is searched literally, i.e. without changes, whereas *stemming* is a form of reducing each word to its word stem. An example for stemming is the word “redness” which is stemmed to “red”.

Through extensive manual inspection of recognized named entities in final results of the original Excerbt, I have found that stemming is a suboptimal solution for the actual problem. In our cases, it is mainly done to reduce plural and genitive forms of words, such as “disease’s” or “diseases” to its original form. However, with our employed solution of the Porter Stemmer algorithm (as part of the Lucene package), both words are stemmed to “diseas” (without the e). Additionally all forms of adjectives and verbs also are stemmed to the same form making an exact identification of the named entity difficult due to various ambiguity problems, for example “diseased”. Another, maybe more understandable example not in the biomedical context would be “apartment” and “apart”. Both terms are stemmed to the same base form but their meaning is entirely different. It is clear that such a stemming would produce a lot of errors.

A better solution than stemming is to only employ a so-called *inflector*. An inflector simply reduces every word to its base form instead of the word stem. The inflector I have implemented now only reduces genitive forms to nominative forms and converts all plurals to singulars. It employs a rule framework for determining the correct base forms depending on the ending, similar to a stemmer but without the overhead to produce correct word stems.

Apart from the issue of stemming, now we do not only have two indices with inverse preprocessing, but also a mixture of the variants case-sensitivity and inflection. Therefore the variants case sensitive and verbatim, case insensitive and verbatim, and case insensitive and inflected have now been additionally implemented. The fourth possible form, case sensitive and inflected, has not been implemented since it does not make sense for any of the entity types in our corpus from a biomedical point of view, but could easily be added. The main reason that before we had only two variants was that the system as

[60] http://lucene.apache.org/, accessed 2012

described in Section 4.1 was not powerful enough to support a 50 - 100% increase in indexing storage and processing requirements. With the novelly implemented cluster architecture, however, this was possible to achieve.

We additionally refrained from using a plain Lucene based solution because of the immense manual overhead in configuring Lucene to behave as expected. Instead, we implemented an approach based on ElasticSearch that takes on the configuration of Lucene and hides it from the end user for easier management. ElasticSearch can also be deployed on a cluster system and is therefore (partially) suited for the high loads necessary to perform Named Entity Recognition. The previous manual sharding is now performed automatically by ES with load balancing and replication.

When performing NER, one is usually interested in the retrieval of *all* hits in the database. A regular search with any search engine usually only gives the first 25 results and offers a paginated solution in order to find the rest. Search hits are ranked by some scoring schema that might be suitable for the use case or not. Hits with higher scores get ranked higher and therefore occur higher in the list. This is, however, a completely undesired behavior when attempting an NER. There we are interested in all results and we do not care about the order the results appear in. No indexing solution, however, is currently especially suited for this task. Various utility functions for ElasticSearch, such as *scrolls* that simulate a paginated version of all results and do not reveal any kind of scores are only of limited use because of a very long runtime.

In order to limit the implications of this problem, we additionally implemented an inverted index data structure in HBase and MapReduce. An *inverted index*[61] is basically a data structure that contains references of words and numbers mapped to documents. For example, the word “is” could be mapped to a list of document ids:

```
is : { 1, 2, 5 }
```

that then denote the documents the word “is” occurs in.

We implemented this, however, not as single words but as an index of *n*-grams. Most entities consist of multiple words that have to occur together within a specific vicinity of each other. For example, the term “Parkinson’s disease” consists of two words. These two words have to occur directly together or with a maximum of one insertion in-between. Also, only one term is not sufficient. Otherwise, they would not represent that entity anymore. If we had implemented the inverted index with words, we would have to search for all constituents of the entity in the inverted index and later overlap those that occur within the same sentence. The problem here is obvious: The huge number of stop words, such as “disease” or “cells” would amount to a drastic increase in runtime compared to when searching them directly. In an inverted index constructed from n-grams, we can search a single multi-word entity all at once without the overhead. The inverted n-gram index is called a sentence token index (STI). This STI contains all n-grams of the entire text resources of Excerbt for $n = \{1, 2, 3, 4, 5\}$. The reason for this is that we can now search for entities that contain up to five words through a direct search in the inverted index and have a constant search time depending on the number of results (instead of a growing access time when using Scrolls or similar techniques). The number five was chosen because of a trade-off between an unlikeliness of finding a hit if the entity consists of multiple words and a

[61] An introduction to inverted index generation and the theory behind is given in Zobel and Moffat (2006)

reasonable processing time. For the majority of the 70 million entities we have in our database, the STI is the preferred (see Section 5.2) way of search. With this approach, a basic lookup table for entities exists and can be used in conjunction with the ES indexing approach. A real-time search feature, for the introduction of new terms into the ontology can now be performed with correct entity preprocessing and subsequent relation generation.

As I have written before, the indexing and NER were originally performed on sentences and PAS. Semantic Role Labeling produces usually a multitude of PAS per sentence, one for each verb. This means that the volume of PAS is much higher in terms of numbers and required disk space than for sentences, resulting in huge indices for PAS and regular-sized indices for sentences. In order to be able to refrain from having to index both, I opted for an implementation of only indexing sentences, but not PAS. Another, maybe more important reason for this is that in a sentence, words can have assigned the same semantic role even if they are disconnected throughout the sentence with other roles in between as explained in Figure 4.5. NER on PAS level might in some cases wrongly detect entities that do not naturally occur as such in the sentence and therefore introduce errors to relation extraction.

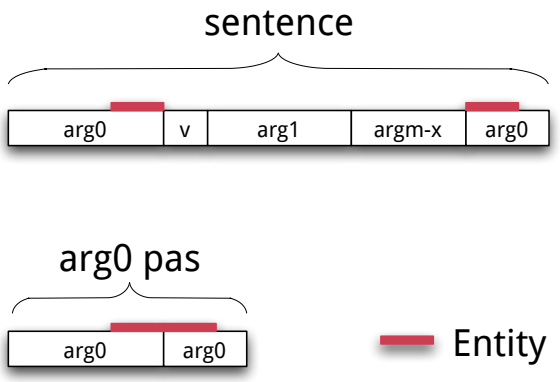


Figure 4.5: NER on sentence level vs. NER on PAS level. A sentence usually consists of multiple different semantic roles. Some roles, such as ARG0 in the example, can occur multiple times in the same sentence. The red bar indicates an entity at the given position. For the PAS generation, roles with the same annotation are concatenated into a single role. This can – in special cases – lead to wrongfully detected entities in the PAS step when compared to the sentence. The two red bars in the sentence would normally not have been detected as a single entity, in the PAS step. However, when performing NER on PAS level, they are.

Indexing with ES supports a technique called *highlighting*. This enables a user to calculate the exact positions of the found entity within the sentence. For the STI we can implemented a similar solution. With the information of the positions of the entities within the sentences, we can now calculate the positions within the PAS as long as we have positional information of single arguments, which we do. Therefore, the indexing and NER have to be performed only once for each sentence, and the occurrences in PAS can be computed upon that information, making the system more efficient and less error-prone.

4.3.1 Percolated Named Entity Recognition

In its current form, all sentences are indexed in the sentence token index STI and in ElasticSearch. Whenever there is a new single document added to the underlying system (via an update or an ad-hoc processing), all 70 million entities have to be searched for occurrence within that single document. In this case, an update of the system only makes sense if enough new data has queued up for processing in order to perform the above explained steps.

The idea was now to reverse this process: What if we could not only search entities in documents, but documents for matching entities? ElasticSearch provides a feature called *percolation* in order to do that. The verb "to percolate" means to "filter gradually through a porous surface or substance" (Stevenson and Lindberg, 2010). Applied to our case, documents are filtered through a sieve that only lets text chunks through that are entities. However, it turned out that this ES feature does not take care of the problem in a way that was useful to us because it only allowed a limited amount of queries and the execution times of a single query were not satisfactory. We therefore decided to implement our own version of percolation.

Percolated named entity recognition is the process of taking a document and analyzing it in a way that entities from a dictionary that are in the document can be identified without having to query each entity against the text. The basic idea is to preprocess the texts into token combinations of n words in such a way, that those tokens can be queried against an index of all entities. In other words, instead of indexing the documents, we now index tokens of entities. Documents are then processed and queried against this entity index. Thereby it is only necessary to perform a low, limited amount of queries in order to identify all contained entities within a single sentence.

The indexing of the entities proceeds as following. Entities are split up into single words (tokens), and each combination of n-tokens (similar to n-grams) is stored in an inverted index. Individual tokens within an n-token are ordered lexicographically ascending. For example, for the two-word entity "Parkinson's disease", the index would then contain: "disease", "Parkinson's", "disease Parkinson's". This ensures that all possible token-combinations for all entities are created and can be queried.

In order to highlight all entities in a sentence, we gradually search all n-tokens of the sentence in the index and mark it if we have a full entity. It is a multi-step process gradually increasing the number of words n per token t. The first step uses all one-tokens (all single words) and queries them separately against the entity index. For a sentence with 12 words this would correspond to 12 queries. In the search phase of step one, all entities that are already complete are identified. Those are entities that consist only of exactly one word and do not occur in any combination within other, multi-word entities. All tokens that do not belong to any entity are removed. Complete tokens are also removed. In the second step, the algorithm computes all two-tokens from the remaining tokens of the sentence and again queries them against the entity-index, and so on (see Algorithm 4.1).

Unfortunately, the process as outlined in Algorithm 4.1 needs to compute

$$N = \sum_{n=1 \to 8} \binom{m}{n} \tag{4.1}$$

Requires: A fast-access entity token index database with individual tokens sorted lexicographically

method NAÏVEPERCOLATEDNER(text)
1: $T \leftarrow$ GENERATENTOKENS(text)
2: **for each** n-token $t \in T$ **do**
3: **if** $t \overset{?}{\in}$ DBTOKENINDEX **then**
4: **return** t
5: **end if**
6: **end for**

Algorithm 4.1: The naïve percolated NER. The naive version of the percolated NER. The algorithm simply generates a list of all n-tokens from the given text and then checks each n-token in a database with all occurring n-tokens in the database created from entity names.

possible n-tokens for a sentence with m words. Each of those n-tokens then needs to be looked up in an index of all possible entities. Since we only compute n-tokens up to eight[62], we are not yet in the complexity range of 2^n possibilities as we would be when computing all possible combinations. Nevertheless this algorithm performs very badly on sentences longer than a few single words. For example, for a sentence with 12 words, for $n = \{1, 2, \ldots, 8\}$, 3796 individual queries would have to be performed against the index. This is better than the 70 million queries but has still a lot of potential for improvement.

I therefore devised a branch-and-bound approach based on the idea of the naïve algorithm (see Algorithm 4.2) that returns the same results but in much less computational time, measurable by the amount of required queries. The general idea is to reduce the needed amount of queries further by only pursuing word token combinations that can actually be part of an entity name and discard tokens that cannot (see Figure 4.6).

Basically, the algorithm starts by generating all possible token combinations for the current n. For $n = 1$, all tokens are queried. It can be considered as some kind of tagging within the sentence of the words that can make up an entity and reducing the possible parts of a sentence until all tagged parts are either untagged or complete entities.

Subsequent combinations for n-tokens with $n \geqslant 2$ have to take into account what valid combinations are. I have briefly mentioned that there can be insertions between the single words of an entity. The number of insertions is limited by the number of words the entity consists of. To account for this, new n-token combinations are combined from the previous ones with a distance between the tokens taken into account. For example, in the string “abcdef”, *ab* and *de* have a distance of 1. If we allow a maximum distance of one for any entity and if *c* is not a valid part of any combination, at most *abd* and *bde* are allowed as next tokens.

To test whether this new algorithm is a significant improvment, I first start with an example and later generalize it with a runtime analysis. Consider the example sentence with 12 words from Figure 4.6. With the assumption that the database contains five entities and three of them are contained in the sentence, we need only 21 queries in total to detect those entities. Compared to the 3769 queries of the naïve approach, this is already a huge improvement. It goes without saying that the precise number of

[62] As the query in this case is the other way around, and the number of entities is much lower than the complete amount of text, we can use a higher number than five.

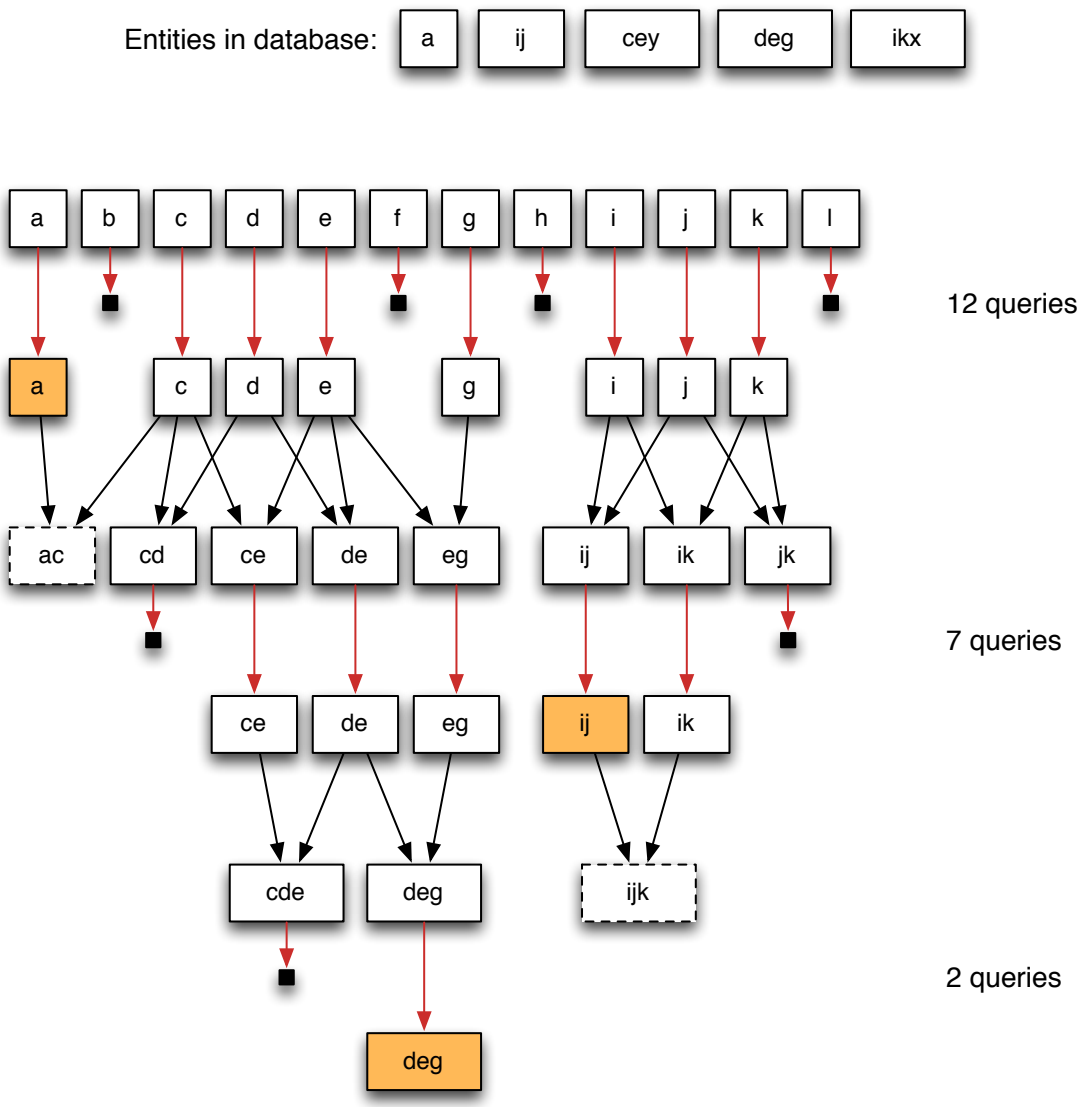


Figure 4.6: EnhancedPercolatedNER explained. We start with a sentence with 12 words *a* to *l*. In the first step we perform for each 1-token (i.e. each word) a query (red arrows) on the database. We find that the words *b*, *f*, *h*, and *l* do not exist as part of an entity. *a* is already a complete entity (marked in orange). Then we compute all 2-tokens out of the existing 1-tokens with a maximum allowed slop of 1, a maximum insertion of one additional word inbetween. The words *ac* (marked dashed) cannot be an entity because the word *a* already denotes a complete entity and is therefore omitted. The remaining 2-tokens are then searched in the database, Word combinations *cd* and *jk* do not exist. *ij* is already a complete entity. Next we compute all 3-tokens out of the existing 2-tokens by creating a union of two elements that have one in common (e.g. *ce* and *de* become *cde*). *ijk* is again omitted because *ij* is already complete. Remaining 3-tokens are searched in the database, *cde* does not exist and *deg* is a full entity. Since only one 3-token is left, we do not have to compute 4-tokens. New 4-tokens would be computed by creating a union of two 3-tokens with two elements in common to correct for slop. We performed a total of 12 + 7 + 2 = 21 queries against the database instead of searching 70 million entities in a single line of text.

Requires: A fast-access index of all possible valid *n*-tokens of all entities, single entity words sorted and n-tokens that are full entities are additionally marked.

method EnhancedPercolatedNER(text)

```
1:  i ← 0
2:  N ← ∅
3:  while true do
4:    i ← i + 1
5:    N ← GenerateNTokens(i, N)                    generate all n-tokens for n = i
6:    if N is not empty then
7:      RemoveImpossibleQueries(N)
8:      for each n-tokens n ∈ N do
9:        Query(n)
10:       CheckIfCompleteEntity(n)
11:     end for
12:   else
13:     break while                                no more combinations possible, finish
14:   end if
15: end while
```

Algorithm 4.2: The enhanced percolated NER The enhanced version of the percolated NER makes use of the fact that the index contains all possible entity tokens and reduces the amount possible queries by taking this into account.

needed queries is highly dependent from the number and composition of individual n-tokens in the entity index and how the sentence is structured. In the best case, where we have no entity at all in the sentence and none of the words can be part of an entity, the number of queries is at most the number of words in the sentence. For the worst case, the number of queries is determined by Equation 4.1. For the average case, however, the number of needed queries should be in the lower region.

In order to test that hypothesis, I selected 20,389 random sentences from the Excerbt corpus and performed percolated NER. For each sentence, I have measured the number of words the sentence contains, how long processing took, how much queries were actually queried and how many entities were found in the sentence. The average sentence length is 23.0 words, which seems to correlate with later analyses of the complete corpus (see Section 5.1), indicating a representative sample. Figure 4.7 shows a plot of the length of the sentences compared to the runtime of the algorithm in milliseconds. Clearly recognizable is a correlation of greater sentence lengths with longer runtimes, suggesting an algorithm complexity of $O(N)$, N being the number of words in the sentence. The mean runtime is 822.7 milliseconds, which means that each sentence was analyzed in less than one second. The average number of queries for a sentence was at 904.6. This number is obviously bigger than the 21 from the previous toy example. One reason is that in a real-world setting, we have to conduct a search for all possible search types (case sensitive, case insensitive, inflected, non-inflected) for all words which significantly increases the number of possible entities. Analyzing a whole document is, however, certainly in the range of possibility now, but not with a regular dictionary-based named entity recognition.

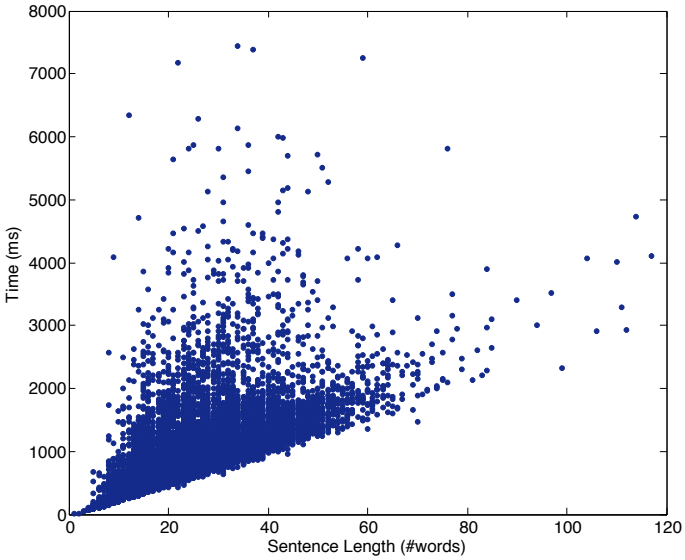


Figure 4.7: Processing time PercolatorNER per number of words in a sentence. For a list of 20,389 randomly chosen sentences from the Excerbt corpus, the PercolatorNER was called and time was measured to complete the queries. X-axis shows the number of words for the sentence and plotted against the measured time. Clearly recognizable is a correlation between sentence length and runtime of the algorithm. The mean sentence length was at 23.0 words and the mean time for a sentence was at 822.7 milliseconds.

The actual number of queries for a particular sentence is determined by the length of the sentence and the number of words that can be part of an entity in the database. Considering that the access to any data entry in an HBase database (where all the relevant queryable data is stored) is in the complexity range of $O(1)$, i.e. constant, the above approximation of $O(N)$ makes sense. An analysis of the number of queries against the runtime shows the same correlation of that in Figure 4.7. This means that the limiting factor is the number of queries that have to be performed throughout the analysis of a sentence. Here, certainly potential for improvement is possible. For example, the algorithm could include knowledge about words or n-tokens that are known to be part of an entity but cannot exist in text with a specific length. In other words it could include information about stop words such as *a* or *the* that can be part of entities, but not be an entity by themselves.

Finally, the algorithm is capable of identifying all entities that are contained in the meta-ontology, if they exist in the investigated text. The basic idea is to prove that if an entity E is present in the meta-ontology and in the text T but was not found by the method, then the respective query K for that entity was not executed. An entity, however, is queried exactly when a combination of n-tokens T_E is generated that contains the constituents of the entity. There are exactly two possibilities where a combination T_E for an entity would not have been generated: First of all, if one of the tokens of the entity do not exist in the text. Second of all, if the n-token index for all entities in the meta-ontology is not complete which would lead to a wrong identification of tokens to use for the next step. If the first point is the case, the entity is by definition not contained in the text and can and should not be found. If the second point is the case, the construction of the token-index of the meta-ontology is not operating correctly. This leads to the conclusion that the presented branch-and-bound algorithm for detecting entities in text is exhaustive, if the token-index of the meta-ontology contains all possible tokens, which is required in Algorithm 4.2.

4.3.2 Summary

To summarize the indexing and named entity recognition steps that were developed so far: We have basically two approaches, the de-novo setup and the incremental update. For an initial setup we have either the ElasticSearch index to perform NER for entities whose words can be freely rearranged and can have inserts, and the STI method for entities that consist of either one single word or whose word order is important. The incremental update is performed via the Percolated NER, which gives here superior performance due to a limited amount of executed queries. For an overview, see Figure 4.8.

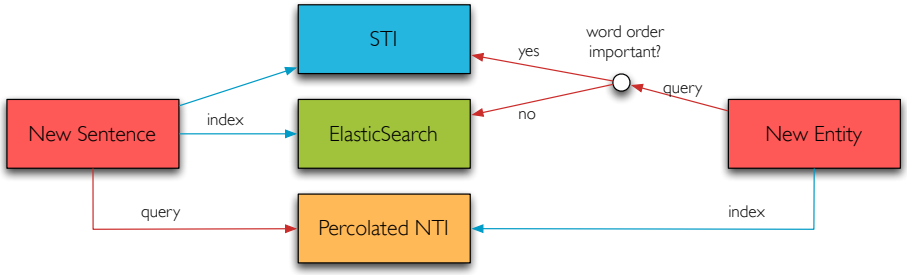


Figure 4.8: Indexing and NER in Excerbt. Three index types exist: The sentence token index (STI), ElasticSearch (ES), and the percolated n-token index (Percolated NTI). A new sentence is indexed in both, the STI and ES and queried against the Percolated NTI. A new entity is queried either against the STI or ES depending on properties of the entity and indexed in the Percolated NTI.

4.4 SRL and Relation Extraction

The original Excerbt extracted a relation from predicate-argument structures if and only if there was an entity each in the `ARG0` and in the `ARG1` annotations and the verb was mapped to a relation type from the manually compiled list. The SRL engine SENNA was used to identify those arguments.

Obviously, this highly restricts the sensitivity of the method. Most of the additional information that is contained in other arguments, such as `ARG2` - `ARG5` and the various modifier arguments, such as time designations, locations, negations, modal verbs, causes, etc. that provide resources for additional relations and various information for contextualization of extracted relations, are ignored. Figure 4.9 shows a sample sentence annotated by humans that presents the amounts of information that could be extracted. With automatic methods, however, this extent is currently impossible to achieve.

Of importance within this context is a publication by Blake (2009) that investigated the way scientists communicate their findings in research articles. They discovered that all scientific statements basically belong to one of five categories: explicit claims, implicit claims, correlations, comparisons, or observations (see Table 4.3).

Blake (2009) performed manual analysis of the different claim categories on a select set of sentences. They annotated 29 full-text articles from Pubmed Central and found that all claims belonging to one of the above categories, 77.11% were explicit claims, 9.23% observations, 5.39 and 5.11 for correlations and comparisons respectively, and only 2.70% were implicit claims. They implemented an automatic

The sentence

The inhibitory action of p27, a cyclin-dependent kinase inhibitor (CDKI), arises from its binding with the cyclin E/Cdk2 complex that results in G(1)-S arrest.

Entities and types

- *p27—individual protein*
- *action of p27—function property*
- *cyclin-dependent kinase inhibitor—protein family or group*
- *CDKI—protein family or group*
- *cyclin E—individual protein*
- *Cdk2—individual protein*
- *cyclin E/Cdk2 complex—protein complex*
- *its—individual protein*

Ontology knowledge about entity types

- *individual protein* $\prec$ *protein* $\prec$ *amino acid* $\prec$ *organic* $\prec$ *compound* $\prec$ *substance* $\prec$ *physical entity* $\prec$ *entity*
- *protein family or group* $\prec$ *protein* $\prec$ *amino acid* $\prec$ *organic* $\prec$ *compound* $\prec$ *substance* $\prec$ *physical entity* $\prec$ *entity*
- *protein complex* $\prec$ *individual protein* $\prec$ *protein* $\prec$ *amino acid* $\prec$ *organic* $\prec$ *compound* $\prec$ *substance* $\prec$ *physical entity* $\prec$ *entity*
- *function property* $\prec$ *property* $\prec$ *entity*

Relationships

- $EQUAL$(*cyclin-dependent kinase inhibitor*, *CDKI*)
- $MEMBER$(*cyclin-dependent kinase inhibitor*, *p27*)
- $ANAPHORA$(*its*, *p27*)
- $CONTAIN$(*cyclin E/Cdk2 complex*, *cyclin E*)
- $CONTAIN$(*cyclin E/Cdk2 complex*, *Cdk2*)
- $CAUSE$($BIND$(*its*, *cyclin E/Cdk2 complex*), *action of p27*)

Ontology knowledge about relationships

- *EQUAL* $\prec$ *equality* $\prec$ *is_a* $\prec$ *relationship*
- *MEMBER* $\prec$ *collection:member* $\prec$ *part_of* $\prec$ *relationship*
- *CONTAIN* $\prec$ *object:component* $\prec$ *part_of* $\prec$ *relationship*
- *CAUSE* $\prec$ *causal* $\prec$ *relationship*
- *BIND* $\prec$ *assembly* $\prec$ *physical* $\prec$ *change* $\prec$ *causal* $\prec$ *relationship*

Text binding, i.e. text used to infer the relationships

- *CONTAIN*: *complex*
- *CAUSE*: *arises from*
- *BIND*: *binding with*

Syntactic annotation

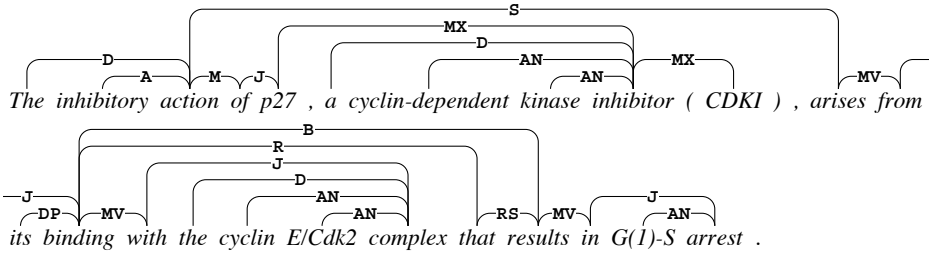


Figure 4.9: What could we theoretically learn from a single sentence? A single sentence contains a lot of information that is inherently clear to a human reader. A machine, however, needs to have a deep understanding of syntax and semantics of a language in order to understand all described facets. No text mining system is yet able to extract as much information as a human, but work is underway. Adopted from Pyysalo et al. (2007).

Category	Concept A	Concept B	Nature of Change	Claim Basis
Explicit claim	req: concept A is agent	req: concept B is object	required	optional
Implicit claim	req: concept A is agent	req: concept B is object	optional	optional
Correlation	required	required	required	optional
Comparison	required	required	required	required
Observation	n/a	required	required	optional

Table 4.3: Types of scientific claims. At most two concepts are involved in a claim, that can either be mandatory (req) or optional. The nature of change gives an indication how the concepts influence each other. The claim basis indicates what the measure of the claim is. Adopted from Blake (2009).

extraction approach for explicit claims based on syntax dependency trees, but were unable to detect the other categories through their approach.

Transferred to our use case with SRL and predicate-argument structures, these categories can be described as follows. An explicit claim is an assertion that is made through a verb. The agent and patient (object) arguments contain a concept and the nature of change between those concept is represented via a verb. In this context (Blake, 2009), a concept is a set of terms that "reflect an abstract of concrete idea within a scientific domain"[63]. An implicit claim does not need an explicit verb, but it needs an agent and an object. An example for this might be a nominalized verb, such as in this example: "LRRK2, a serine/threonine kinase, . . . ". in this case there is no explicit verb that connects LRRK2 with kinase but the interjection implicitly states that the two concepts are related through an is-a relationship. A correlation requires two concepts that somehow are involved in the same process but do not necessarily influence each other. In fact there is usually no mention whether concept A causes concept B or vice versa, or a third concept influences both. Comparisons simply state a comparison between two terms. Usually those can be detected through the presence of comparatives, such as higher, lower, more, or less. Finally, an observation is a simple statement that a change has occurred, but gives no information by whom this happened or what the cause of the change was. A simple example would be "excretion rates increased".

In collaboration with Barbara Hummel, a master's student under my supervision (Hummel, 2011), we did extensive analysis of the sentence corpus of Excerbt with regards to their semantic roles in accordance with the above described claim categories. We discovered that explicit claims are already covered in part by our approach of using entities in the `ARG0` and `ARG1` roles within a sentence. Correlations and comparisons can easily and with great confidence be extracted from semantic role labels. They mainly depend on the presence of specific keywords in specific arguments within the sentence. For a list of all generated rules, see the Appendix on page 157.

Observation claims can also be detected but with a limited performance due to many false positives. The general idea here is that we only have a single concept and a change term that we could simply search for entities contained in the `ARG0` role label and use verbs that represent a change.

The detection of implicit claims is unfortunately not that straightforward. We therefore put specific focus on the time designation and location arguments (`ARGM-TMP`, `ARGM-LOC`), verb modifiers such as `ARGM-MOD`, and negated sentences containing `ARGM-NEG` in order to extract a context in which

[63] Often the word "concept" is simply used to express a general idea. In this case it is the actual representation of said idea with words.

specific relations take place. The basic question was, how this additional information could be used to significantly improve detected relations with regards to verb qualifiers, and find more relations that might be hidden in other arguments ARG2, ARG3, etc.

These results have been incorporated into the new system in two forms: a new set of rules that can detect specific types of relations (such as the correlation) and a contextualization of existing rules. The relation data in Excerbt can therefore be annotated whether it is a negation or a hypothetical statement, and the additional relation types correlation and localization were implemented (Hummel, 2011).

In collaboration with another master's student of mine, Robert Strache (Strache, 2012), we additionally discovered that relations could be extracted from the causative and temporal arguments of PAS. These arguments can, under certain circumstances contain a useful reference to the source of an interaction. In the example in Figure 4.10, the part of the sentence after "after" is correctly identified as a temporal relation to the PAS. "Dendritic cells" is either the agent or the patient of the verb in this case but can be used in conjunction with the contents of the temporal argument in order to produce a valid relation.

Figure 4.10: Temporal arguments as a basis for relations. Adapted from Strache (2012).

With these two rules, two relations can be extracted from this sentence that would have been missed entirely otherwise:

Mycobacterium tuberculosis	activates	dendritic cells
Mycobacterium tuberculosis	matures	dendritic cells

A full list of all rules can be found in the Appendix on page 157.

I have earlier mentioned that PAS might be included in each other. Consider a subordinate clause that is completely included in a specific role of a main clause. At this time, we cannot make the connection between these two resulting PAS, resulting in a lot of false positive extracted relations from the main clause PAS with entities that are actually part of a role of the subordinate clause. For this reason, we limit the detection of named entities in the various roles of PAS to part up to the first occurrence of a verb as exemplified in Figure 4.11 (Strache, 2012).

Figure 4.11: Limiting arguments up to first verb. Adapted from Strache (2012).

Normally, this sentence in the original unlimited form would have given false positive relations with "apical tuft" and "microvilli". When looking at the sentence, however, it becomes evident that the actin filament does not actually relate to the entities after the second verb "stain" and therefore the ARG1 in this case can be limited up to the first verb.

Taken together, these improvements in the semantic role labeling and relation extraction steps of the Excerbt text mining system greatly improved sensitivity of the approach.

4.5 Efficient Knowledge Representation in Column-Oriented Databases

The previous version of Excerbt stored data in a PostgreSQL database. As previously argued, this kind of databases (relational DBs) were not sufficiently able to keep up with increaing data volumes. Since I have decided to employ the column-oriented HBase database that is compatible to the computing framework Hadoop MapReduce, a new type of data representation had to be developed. A standard SQL data schema usually is very inefficient in a column-oriented database.

Storing knowledge on a computer in a way that the computer can (based on useful algorithms) semi-automatically make inferences from it is an important task for any knowledge extracting system. Excerbt is now a text mining system that extracts facts and assertions in a large scale from massive amounts of literature data. For the various advantages as outlined in the preliminaries, we opted for Topic Maps to store extracted knowledge (see Section 2.6). The specific question now was, how to transform the topic maps approach to a level capable of handling big data. As data in Excerbt is processed and stored through HBase tables, I had to come up with an HBase schema that allowed the batch-processing of large data volumes as well as the quick performing ad-hoc queries to the database

Tables in column-oriented databases can be categorized into two types: *tall-narrow* tables that contain an extremely large amount of rows with individual rows only containing little data, and *flat-wide* tables where each row contains massive data but the total number of rows is limited (George, 2011a). HBase's approach to data handling basically poses the statement "billions of rows times millions of columns"[64]. This statement gives the hint that HBase currently favors the tall-narrow approach with tables that have many rows and a magnitude less columns within those rows, however, work is underway to support flat-wide tables in HBase as well. As we will see shortly, this has an impact on table design.

An implementation for the contents of individual documents, such as the raw data, authors, journal, publication date info, etc. is straightforward. Likewise for sentences and PAS. In order to enable a better usage of the table structure in HBase, it is common to write data denormalized and redundant within the tables. A table in HBase has only the row key to access a specific row. Rows are sorted lexicographically and can be accessed in constant time. If we'd like to search for a particular column value within a table, we would either have to perform a scan or have some kind of index table in HBase or elsewhere that allows us to retrieve corresponding row IDs. Therefore, the information in tables that save the hits of named entity recognition is additionally stored in reversed format in the sentences and pas tables.

Just to explain the approach in a bit more detail, the row key of the table that stores information on found hits in the NER step is constructed the following way (`cs_v` stands for case-sensitive, verbatim)

```
<entity name>|<search type>|<document id>
```

[64] http://hbase.apache.org/, accessed 2012-07-05

```
e.g.: LRRK2|cs_v|pubmed:15680455
```

The reason is that if the row key would only consist of the entity name and all other information would be stored in the row's columns, the row would become too wide for certain hub entities and neither HBase nor MapReduce would be able to process that row. Therefore, we added additional information such as the search type (whether it was searched case-sensitive or verbatim etc.) and the document identifier. So a single row contains all occurrences of the named entity within a single document. Through the access of table contents by scan, this is not a problem. A simple scan for all rows that start with a given name yields all desired columns and has a linear run-time with the number of rows to be retrieved.

Topic Maps, as introduced in Section 2.6, are usually complicated to efficiently store on a computer because of the hypergraph relations that can be posed between any number of topics and the resulting high degree of connections between data points. There are various relational database schemas in existence that attempt to model topic maps onto tables. For my diploma thesis I created such a schema for MySQL and PostgreSQL databases (Wachinger, 2009). Because we use the column-oriented database HBase to store all our data, we had to devise a mechanism of storing Topic Maps in an efficient way in HBase.

The core of the Topic Maps database schema consist of two tables for topics and associations between those topics (see Figure 4.12), as expected. The topics table contains all information about a single topic with references to the keys of the associations. This is straightforward and would have been implemented in a similar way in other database systems.

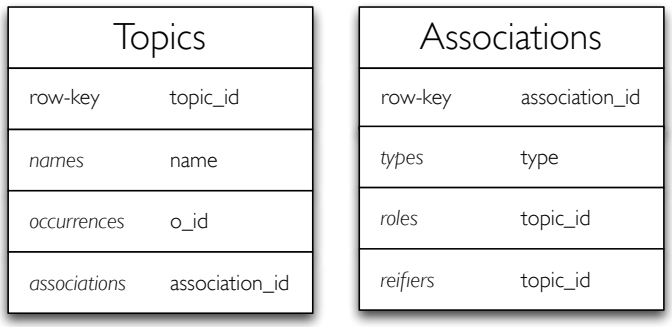


Figure 4.12: HBase schema for topics and associations. Row keys are the identifier for the respective table (see text). The topics table has basically three column families: names, occurrences, and associations, which have a reference to the respective component as the key and a stub object which contains the most important parameters as the value. Equivalent definition for the associations table, which has references to roles (the participating topics of the association), types and reifier objects.

The key for a topic is constructed from the simple label of the topic together with the type. In our cases it is the name of the entity and the respective type. Stored in this way, the topics table can be search by a Get operation. Scans in this scenario do not make much sense because we are usually either interested in a single topic or in a range of topics with similar properties such as a combined neighborhood, but usually not an alphabetical listing of entity names. The key for the topics looks like this:

```
<entity name>|<type>
```

```
e.g.: LRRK2|gene
```

The key for an association is made up of the participating topic identifiers and their type of interaction in order to have two entry points: a search for a specific topic can transpire through the topics table and a search for interactions of a specific topic happens through querying the associations table. We are also able to store hypergraphs as the topic map specification imposes. In that case, an association has multiple sources and/or targets when speaking in graph notation. The individual sources and individual targets are then sorted lexicographically for the construction of the key in order to enable a consistent key generation and identification. The sources and targets are hashed by a specific function to minimize the length of the key as in this example:

```
<type>|hash:<source1>+<source2>+...||...+<target1>+<target2>
e.g.: activates|6d45e8b33
```

The official ISO specification for Topic Maps includes more than described here or in the preliminaries. Some particularly difficult parts of the specification have not been implemented in our solution. These include variants and scopes[65] that are not needed in our use case. Therefore, not every topic map can be stored in our database. But everything that can be saved in the schema merits to be called a topic map. We basically have developed a topic maps engine on top of HBase, which implements part of the official topic maps API[66].

The work introduced here has been presented at the TMRA conference in 2010 (Sixth International Conference on Topic Maps Research and Applications): “Scaling topic maps to billions of topics and associations” (Wachinger and Stümpflen, 2010).

4.6 N-Gram Trends

N-Grams are, as introduced in the preliminaries (Section 2.5.5) often used to access statistical patterns in text. They can be considered a useful tool when implementing statistical text mining because they give frequency distributions for all phrases within all sentences and documents. The frequencies of different n-grams can be compared to each other and give insights into the the evolution of certain concepts and phrases.

A first approach similar to the one described here was published be Michel et al. (2011). The authors took the digitized versions of approximately 5.2 million books from the Google Books project[67], corresponding to approximately 4% of all books ever published, and performed an n-gram frequency analysis. Michel et al. (2011) call their approach *culturomics*. It was picked up later by Leetaru (2011) who performed a comparably analysis for news articles.

Having sufficient resources due to Excerbt’s new architecture, I have implemented a similar approach on the corpus of Pubmed MEDLINE and Pubmed Central documents. I computed all n-grams for $n = \{1, 2, \ldots 8\}$ and stored them in the database in a way that makes retrieval of the frequency distributions

[65] see Garshol and Moore (2008) for details
[66] http://www.tmapi.org/, accessed 2012-08
[67] http://books.google.com/, accessed 2012-08-09

for individual terms and phrases very efficient. The frequencies are stored separately for each month for the publications. Absolute frequencies of occurrences of n-grams obviously are not comparable to each other. The frequencies are therefore normalized according to two criteria: number of documents in that month and number of sentences in that month.

A retrieval system for N-Gram Trends has been implemented for the web interface of Excerbt (see Figure 4.15 in Section 4.7.1). I have performed multiple analyses on the N-Gram corpus in Chapter 5.

4.7 User Access

Obviously any text mining system requires a sophisticated user access. In the following, a short overview over the various methods to access the Excerbt Text Mining System from a client side is given. The basic idea is to provide RESTful web services for scripting access and a web interface for normal use in order to access the big data structures of the Excerbt topic map.

At the core of the user interface was the question, how a user might be able to gain a maximum of information out of the system. The fundamental point here was the creation of a query language specifically designed for the Excerbt system. The interface's design and usability are based on innovative design principles that simplify the possibilities to work with a web-based system. On the server side, user access is handled by an application server providing the web services written in Java. The client side is completely implemented in HTML, CSS, and JavaScript.

4.7.1 Web Interface

Excerbt is outfitted with a sophisticated, clean web interface, accessible through http://mips.helmholtz-muenchen.de/excerbt. It is the most advanced method for retrieval of the extracted knowledge in Excerbt. The key idea behind the web interface was to provide biologists a way to access relevant information in as easy method as possible.

Figure 4.13 shows a screenshot of the system with a sample query and results. At the heart of it is the query bar where the user can enter arbitrary queries in a specially designed human-readable query language. In this case, the query is "GENE LRRK2" which returns all the knowledge found for the particular gene LRRK2. The output is in tabular form as it is easy to navigate through the data this way. A relation in Excerbt is, to bring it back to mind, always an edge in a network with multiple identifiers: a source type, a source name, a relation type, a target type, a target name, and finally some evidences that show the literature citations for this particular interaction. Both, the query language and the tabular view are oriented by how to access graphical network structures in an efficient way by still maintaining a general view. This means that everything that can be displayed within a column (except the evidences) can be specified in the query language, allowing a specification of the query to the exact information that the user is looking for. In order to get the highlighted evidences as in the figure directly, a user would have to enter

```
GENE LRRK2 ACTIVATES PHENOTYPE "Parkinson's Disease"
```

as the query.

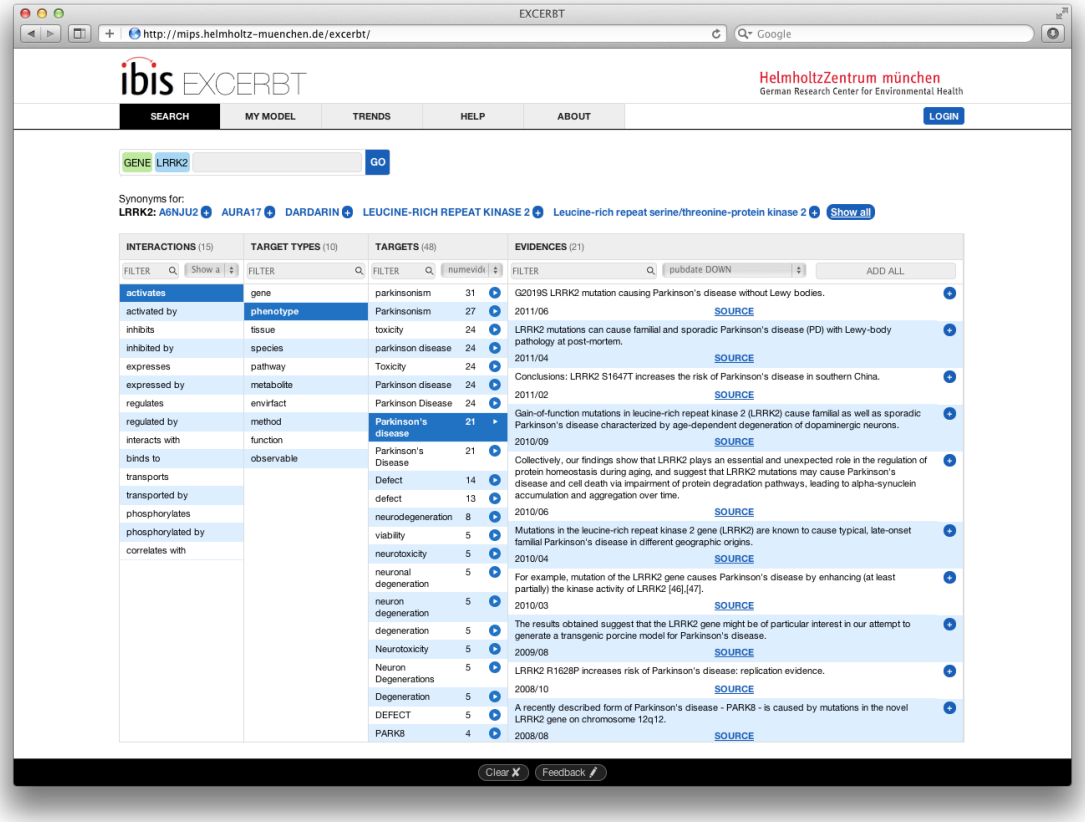


Figure 4.13: Search module of the Excerbt web interface. Displayed are the results for the query "GENE LRRK2". Additionally, synonyms to the query LRRK2 are given. The blue-white right arrow in the Targets column can be used to start a new query from this target entity. Entries in columns are ordered according to certain criteria and can be filtered by the user.

A distinctive feature of the interface is the declaration of synonymous ("Show all" button in Figure 4.13 and Figure 4.16) names to the entered query and the possibility to include additional names into the search by simply clicking on the item or entering them additionally to the query.

The interface provides several additional modules:

1. *Search*: This module is for the regular relation search through the query language (Section 4.7.2). Users can browse in the mentioned tabular view through the data (see Figure 4.13) and inspect publications or add individual relations to a model.

2. *My Model*: here, a user can make use of the small + buttons next to the evidences and create a graphical model of the particular entities and evidences that were selected in the Search tab (see Figure 4.14). The model can be downloaded as native JSON format and is stored locally within the browser for later use. A GraphML and CSV download option will be included in a future version.

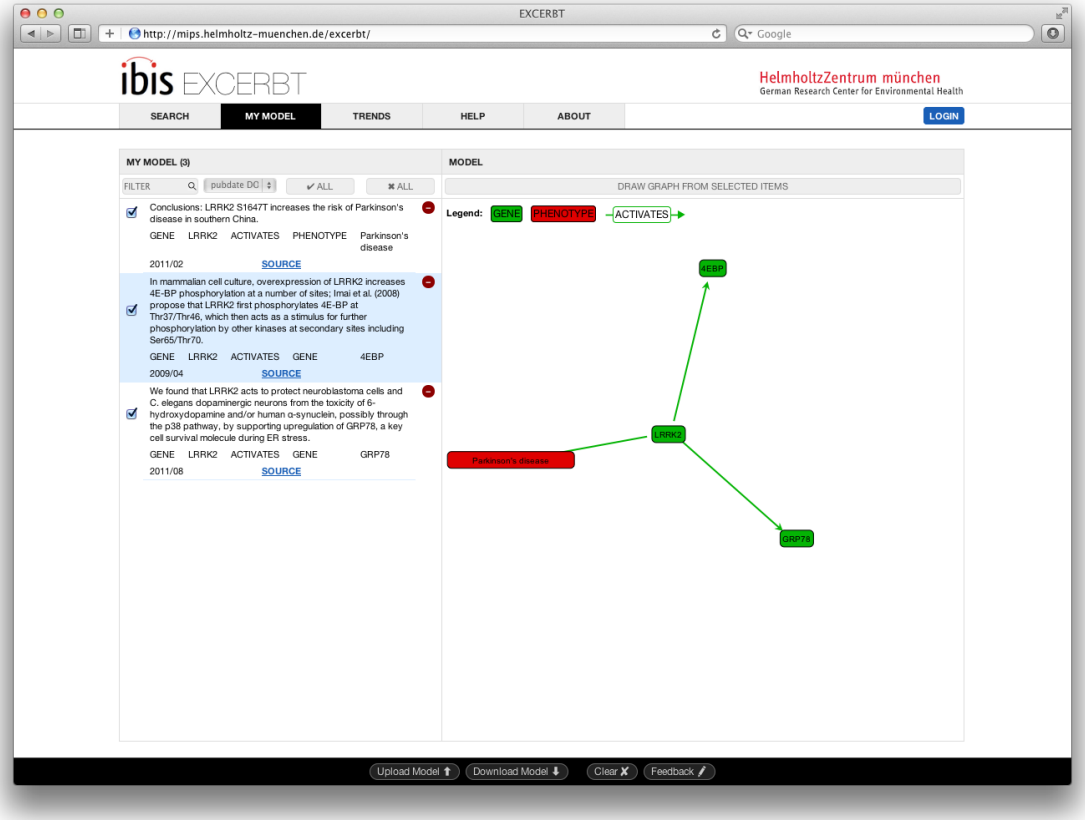


Figure 4.14: My Model module of the Excerbt web interface. In this example, I have created a model of four entities and some of their interactions. The left column shows the evidences that support those relations. Phenotypes are displayed in red and genes are displayed in green.

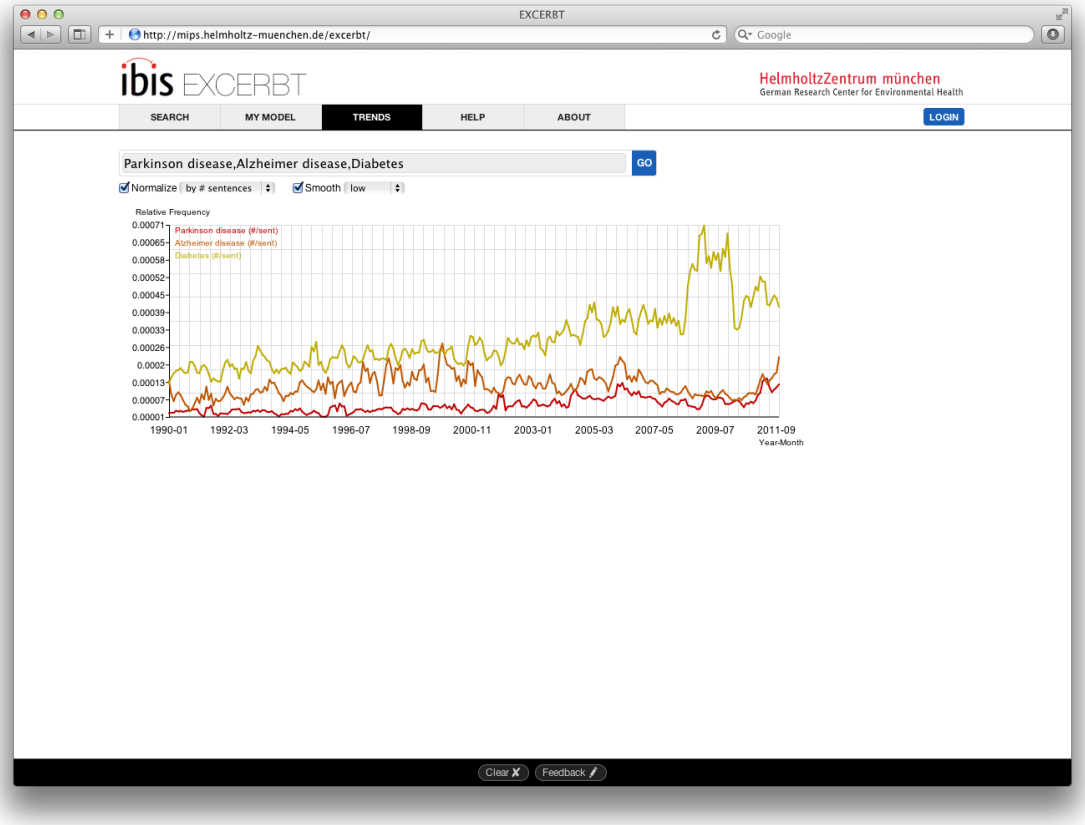


Figure 4.15: Trends module of the Excerbt web interface. Here an example comparison between the three diseases Parkinson's disease, Alzheimer's disease and diabetes according to frequency in publications is given. While it is evident that the interest in those diseases rises in the last couple of years, the trend for Diabetes seems to be more steady whereas real interest for the other diseases only came up since around 2010.

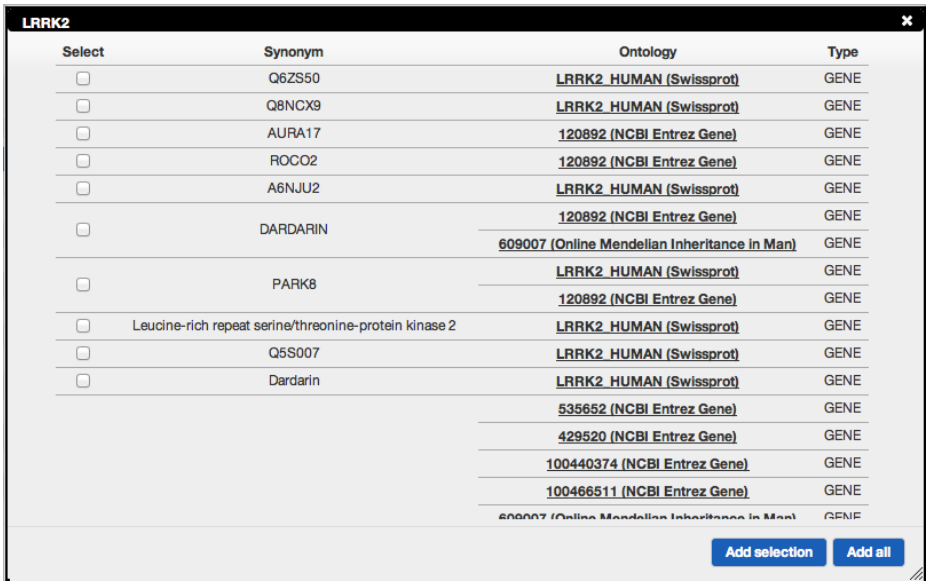


Figure 4.16: Dialog to choose from synonyms for LRRK2. In Figure 4.13 a "Show all" button is displayed that leads to this popup where a user can select additional synonyms for the query.

3. *Trends*: This is the query interface for the n-gram-based trends in biomedical publications (see Figure 4.15). Here, the user can search for arbitrary terms (with the help of an autocomplete functionality) that might occur in the corpus.

While the first two (Search and My Model) can only be used in conjunction, the third (Trends) is independent from the other two and allows a statistical view on the frequencies of certain n-grams within the publications.

Communication between the web interface and the underlying database is done through the API of a multitude of different RESTful Web Services. The API can be accessed directly by power-users but is not intended to be used by biology-oriented scientists. A description of all those web services would go beyond the scope of this thesis. The central entry point to the system is through the query language. The Web Interface is basically a visualization engine for the output of that particular query language and other web services.

4.7.2 The Excerbt Query Language

The Excerbt query language (EQL) has been carefully designed to provide a natural language access to the data stored in Excerbt. As all data is stored as relations, two nodes with names and types, and an edge with a type between them, the query language reflects that notion and allows a user to formulate queries for a variety of results. Basically, the following two modes of operation are supported: direct relation search and indirect relation search.

Whether one is interested in the neighbors of a specific entity for specific types, or the relation types between two entities or basically anything that is known for a specific entity, the EQL supports it. It

needs at least the name of an entity and offers in the “convenient table view” results and alternatives to the query.

The basic query is simply the name of an entity, for example `LRRK2`. If one is interested in a specified entity of a particular type, he/she can write the type before the name: `GENE LRRK2`. This unit can be extended further by specifying more types or more names, such as `GENE DOMAIN LRRK2` or `GENE LARRK2 DARDARIN`. Then, every valid result for every combination of types and names is retrieved. An entity can also be queried through the use of its original database identifier, for example an EntrezGene ID, such as `ENTREZ_GENE 120892` (which is the human gene LRRK2). Internally this is translated into a synonym resolution and separate searches for all aliases.

Additionally, the EQL allows to specify a relation type. The sample query can be enhanced to look for all targets that are regulated by LRRK2: `GENE LRRK2 ACTIVATES`. After the relation type, another type-name unit is allowed or simply a single type, such as `GENE LRRK2 ACTIVATES PHENOTYPE`.

The specified units in the query and the columns in the result correspond to each other. The basic units are: source, relation type, and target. Depending on what is specified in the query, it will not occur in the result and vice versa. Everything that can be displayed in a column (except evidences), can be specified in the query. A complete query for evidences that support a regulating influence of LRRK2 to Parkinson’s disease might look like this:

```
GENE LRRK2 REGULATES PHENOTYPE "Parkinson’s disease"
```

It results in a list of evidences supporting that relation.

Additionally to the regular entity types, such as gene, phenotype, metabolite, etc. and relation types, such as activates, inhibited by, or regulates, etc. an `ANY` type is supported: `ANY_TYPE` and `ANY_INTERACTION`. With these keywords, the EQL is instructed to ignore the type and simply return everything. The difference between `ANY` and nothing is that when no types are given in the query, the results offer potential intermediate types, while with `ANY` they are combined into a single result (in other words, less columns in the output).

For example, the query

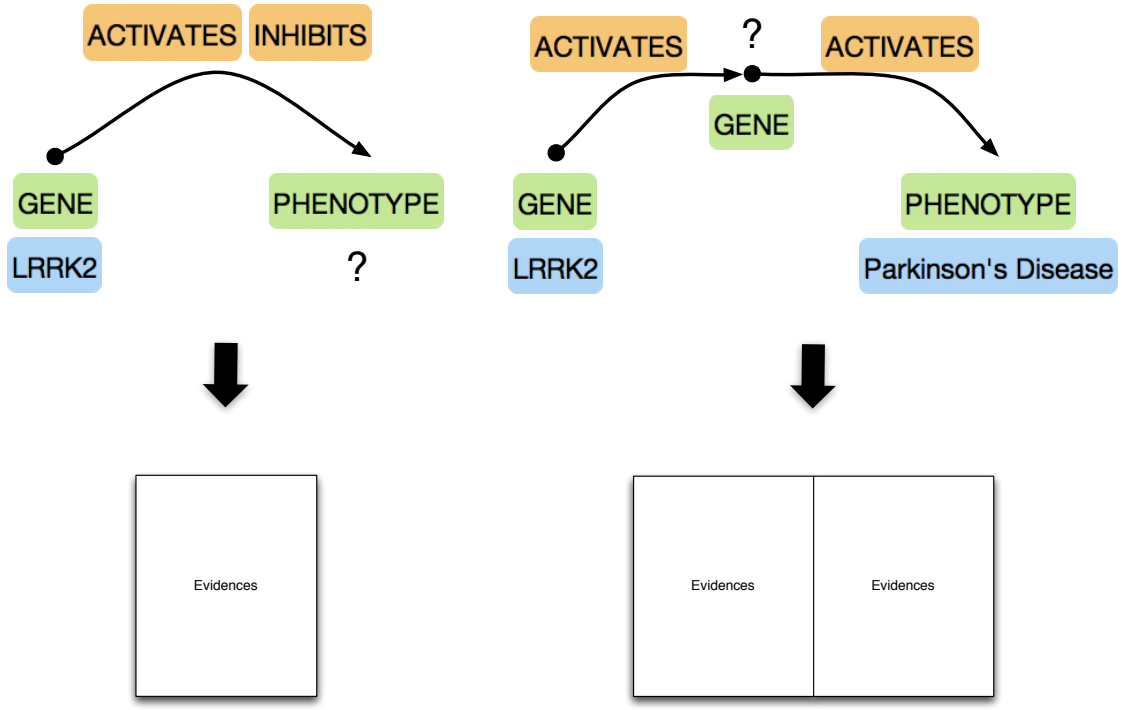
```
ANY_TYPE LRRK2 ANY_INTERACTION ANY_TYPE "Parkinson’s disease"
```

results in all evidences that can be posed between any types of the two entities in any relationship. Basically it gives every edge found by the system between them. A useful shortcut for this query can be performed with the keyword `AND`: `LRRK2 AND "Parkinson’s disease"`. This is again internally translated.

Additionally to this direct relation and evidence search, Excerbt supports a novelty (to my knowledge) in the field of text mining: a support for submitting *indirect queries*. An indirect query is one that specifies two endpoints but instead of finding evidences between them, it finds additional entities that have relations with both terms. For example, if I have a drug, such as Imatinib, and have discovered that it can do something about Diabetes, but I’d like to know more about the involved processes, I could search for intermediate genes that have connections to both, the drug and the phenotype and investigate those further. The query would look something like this:

```
METABOLITE Imatinib INDIRECT PHENOTYPE Diabetes VIA GENE
```

The web interface supports this type of query similar to a direct query. First, a list of intermediate entities is presented. Upon clicking on them we can further restrict the involved relation types and e.g. look for relations that are orthodromic ($\rightarrow \rightarrow$ or $\leftarrow \leftarrow$) or antidromic ($\rightarrow \leftarrow$ or $\leftarrow \rightarrow$) depending on what we want. For inclusion in a model, evidences for both interactions are displayed in parallel.



(a) relation / event queries

(b) indirect queries

Figure 4.17: Standard and indirect queries in Excerbt. (a) The basic aim for relation/event queries is to search for evidences for a particular interaction or the interaction partners of a particular entity. (b) In indirect queries, one is interested in entities (and evidences) that mediate a given known (possibly proven) relation. Since this outputs two relations, we also have two different kinds of evidences.

Don Swanson argues in his paper where he introduced the ABC-model (Swanson, 1986) that if two relations "A causes B" and "B causes C" are known, one might argue that also "A causes C" is true. In the web interface, one could simulate such a query by searching for all neighbors of the neighbors' of the query node. Excerbt's indirect query method basically goes the other way around and finds for a given relation "A causes C" all relations to an intermediate B that might explain the interaction.

Figure 4.17 exemplifies the direct and indirect queries graphically. The main difference is that for direct queries we search for evidences for a single interaction. For indirect queries, we search for an entity involved in both, source and target, and resulting intermediate evidences.

4.8 Summary

In this chapter I have introduced the technological advancements to the text mining system Excerbt that I have developed. The main focus of the Excerbt text mining system is now the capability to handle large-scale exponentially increasing data. The topic maps schema for column-oriented databases that I have developed can be considered a viable approach for such systems in that it supports concurrent access by thousands of people at the same time due to the distributed nature of HBase and a sophisticated schema creation. The web interface is implemented as a stand-alone application server that can be started on multiple servers and theoretically load-balanced through a single web address.

Contextualization of relations is implemented to a large part. What is left is a useful integration into the query language in order to discriminate usefully between negated and hypothetical interactions. The various rules for relation extraction can definitely be enhanced upon. The next steps within this projects are to define a domain-specific language (DSL)[68] in order to simplify the extraction of knowledge via rules defined in this language.

We also want to implement a graph-exploratory view into the web interface. Starting with arbitrary nodes (entities), the user should be able to create a model by clicking on nodes and adding edges on the fly. Until now, however, we have not found an efficient implementation to view end edit networks in a web browser.

Figure 4.18 shows a summary of the steps involved in text mining with Excerbt. Natural language processing methods, and relation extraction are features necessary for the system to build a graph structure that is then stored in the HBase topic map store. This then enables the exploration of the graph through the web interface and question answering modules. Hypothesis generation of statistical analyses will now be covered in the next two chapters.

[68] http://en.wikipedia.org/wiki/Domain-specific_language, accessed 2012-07-05

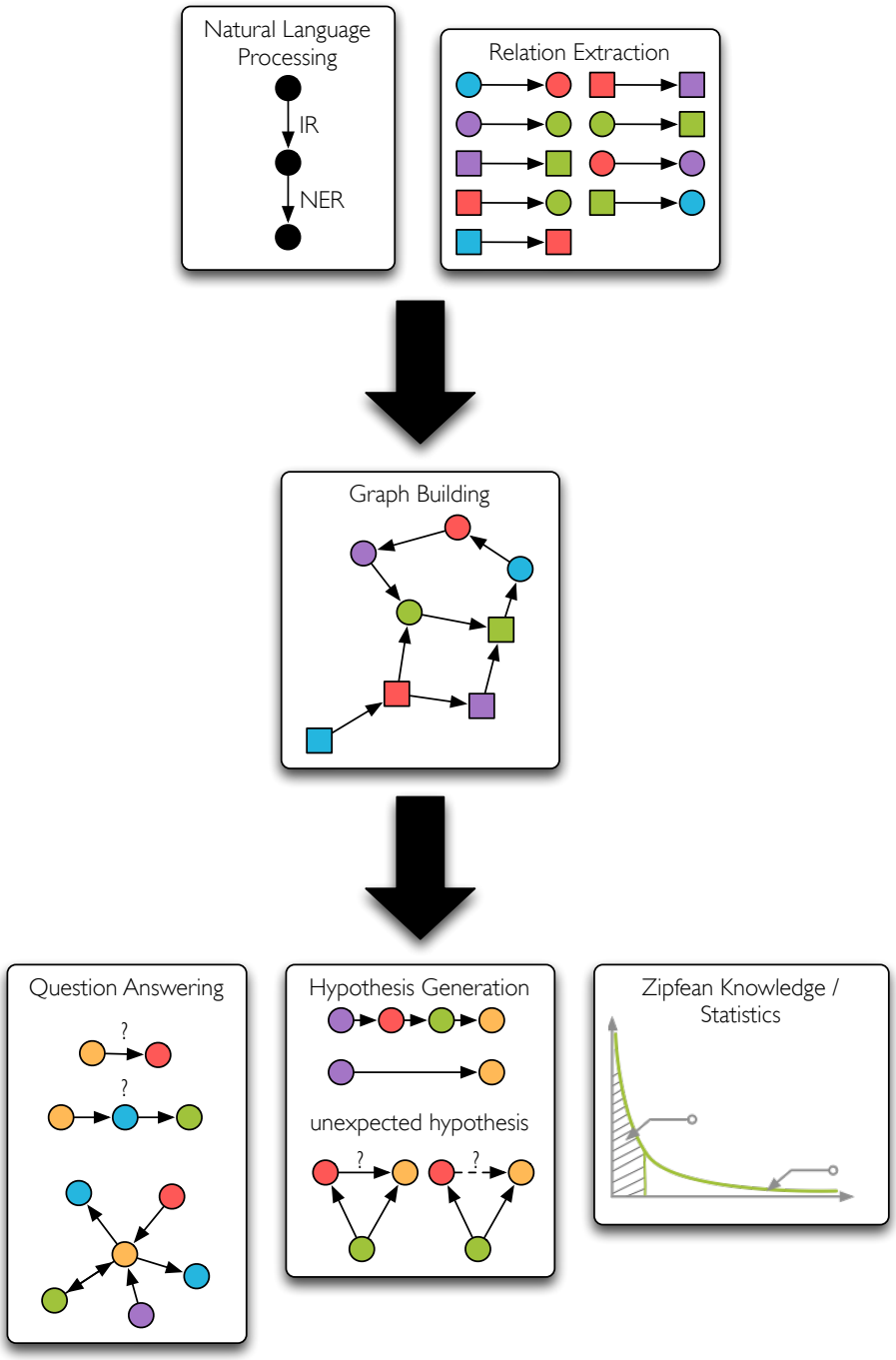


Figure 4.18: Text mining with Excerbt. Natural Language Processing and Relation Extraction represent the building blocks for text mining systems. Core of Excerbt is a graph representation of extracted concepts and assertions. Emanating from that graph, individual user question answering, hypothesis generation and statistical knowledge mining are possible applications.

5 A Close Examination of Excerbt's Data

This chapter serves mainly two purposes: First of all it should demonstrate the abilities that are now enabled through the choices for architecture and design of the novel Excerbt. Fast ad-hoc analyses on big data and a flexibility in what to include in those data are major points here. With this new architecture, a processing of 230 million predicate argument structures from documents into relations is now possible in the time range of a few days. While the initial generation of data is an important step in analyzing the capabilities of the system is how effectively it works with the generated data for statistical and other analyses. Sections 5.1 to 5.3 give a statistical overview of the generated data. Second of all, this chapter aims to provide an evaluation of the relation extraction performance. Section 5.5 gives an overview of evaluating text mining systems and exemplifies data- and user-driven evaluations on Excerbt's extracted relations.

I will first start with analyzing the Excerbt data corpus from ground up, i.e. I will start with documents, continue with sentences and PAS and finally analyze extracted relations. Entities will be handled separately. Unless otherwise noted, the data is as at July 2012. For different analyses I have used data at different points in time. As the data, however, only changed quantitatively but not qualitatively during the last year, this makes no difference to the overall results.

An additional general point is that due to the architectural design of Excerbt with an emphasis on big data applicability, most computations took only a couple of minutes to compute on the data corpus. In contrast to this, similar computations with the previous relational model would be in the range of at least a couple of hours, which made fast ad-hoc analyses impossible.

5.1 Documents, Sentences and Predicate-Argument-Structures

The new Excerbt contains a total of 21.8 million documents consisting of 21.4 million citations from Pubmed MEDLINE and 0.4 million articles from the open-access part of Pubmed Central, which means that about 1.8% of the documents are from Pubmed Central. The number of documents increased exponentially over the last decades and are likely to continue so (Figure 1.1). I estimated that the number of published articles indexed in Pubmed will increase to approximately 38 million within the next three to eight years (2015-2020)[69], which means the data from now will nearly double by 2020. Obviously, the exponential increase in published literature cannot continue unstopped. A plateau in the increase of new publications indexed to Pubmed will likely arrive within the next couple of years due

[69] Estimation was done via regular exponential curve fitting $f(x) = a \exp(bx)$ on the data of Figure 1.1. a was estimated at 3.089×10^{-46} and b at 0.06052.

to a saturated amount of research that can be done. An exact prediction when that plateau will arise cannot be given. However, Pubmed Central and other resources increasingly open up the access to full-text publications. Since abstracts are of considerably less volume than full-text articles (as we will see later), the plateau in increased numbers of documents will most likely only marginally influence the increase in total data volume.

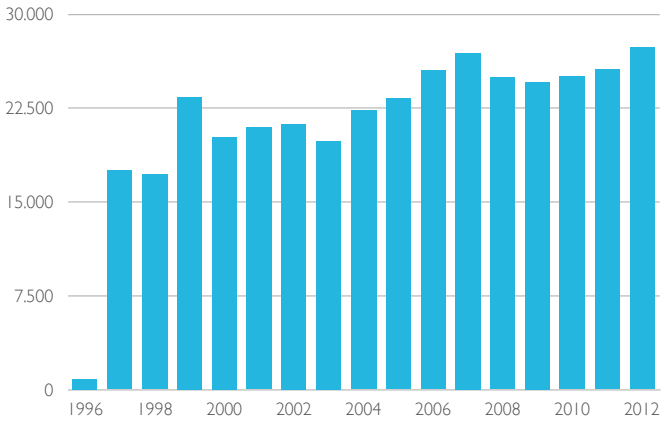


Figure 5.1: Average length of Pubmed Central articles over the years in characters. Clearly recognizable is a slight increase in article length, with local peaks at 1999, 2007 and 2012. The value for the year 1996 can be attributed to a glitch in data or insufficient information. PMC started in early 2000[70]. Data as at July 2012.

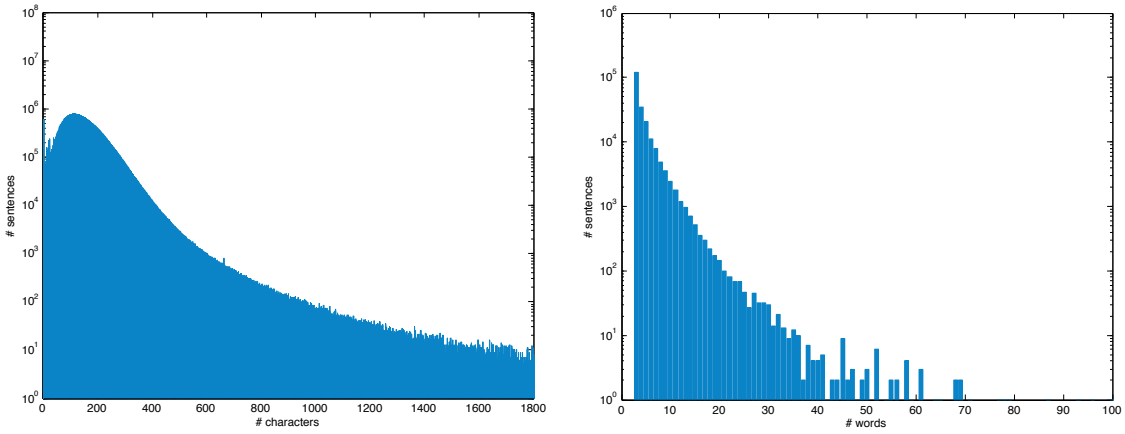
According to the text mining pipeline from Section 3.3 these current 21.8 million documents can be split into 169.9 million sentences (109.5 Pubmed MEDLINE, 60.4 Pubmed Central). The percentage of sentences from Pubmed Central rises now to a value of 35.5%, which demonstrates the point from above. On average, a single document contains about 6.3 sentences. Not all citations within MEDLINE have an abstract, therefore the average number of sentences for those with only an abstract is at approximately 8.9 sentences per document and for PubmedCentral articles at 42.7. Interestingly, the length of full-text articles seems to slightly increase over the last few years (Figure 5.1). A reason could be the general trend to publish and write more that can be observed on nearly every level (Internet, Social Media, etc.).

As a side-note, it is necessary to point out that this development has also other implications: As full-text articles generally contain more information than simply the abstract which usually only highlights the best findings (35.5% of sentences, but only 1.8% of documents), articles in open-access journals that are indexed in Pubmed Central have generally a better chance of being identified as relevant. In other words: publishing in an open-access journal also increases the availability of the resource for automated methods and a chance of being included in automatically generated models and not lost in the literature.

The average length of a sentence is 21.5 words or 145.3 characters respectively (Figure 5.2).

Semantic role labeling yielded around 227 million PAS for all sentences. For each verb in a sentence, SRL produces one PAS, meaning that any sentence in the corpus contains 1.3 verbs on average, including

[70] http://www.ncbi.nlm.nih.gov/pmc/about/intro/, accessed 2012-11-01



(a) (b)

Figure 5.2: Sentence length histograms for characters (a) and words (b). Most of the sentences have between 100 and 200 characters. The average number of characters in a sentence is 145.3. The average number of words is 21.5. The maximum number of words seems to be limited at around 70. Sentences with that high a number of words probably are either simple listings of gene names used in studies, a list of author names participating in a study or errors in the detection of sentence boundaries. The exact numbers for each of those categories are unknown. Data as at 31/12/2011.

those without verbs (title sentences). Figure 5.3 shows that most sentences have between one and three PAS with a quick decline for the number of sentences with multiple PAS. Sentences with as much as 35 PAS likely seem to be parsing errors.

5.1.1 Negations and Hypotheses

Within the scope of a master's thesis of a student under my supervision, Barbara Hummel (Hummel, 2011), we have conducted an analysis of negated and hypothetical sentences in Excerbt's corpus.

A sentence states a hypothesis, if the assertion is not a fact but relativized by additional words such as *probably*, *presumably*, *may*, *could*, or similar. In order to get a feeling, how many sentences contain hypotheses, we conducted a frequency analysis of the modal verbs in PAS (Table 5.1). In the English language, nine modal verbs exist: can, may, could, should, will, would, might, must, and shall. Additionally, there are phrasal modals, such as *going to* and *have to*, which are not included in our analysis. Not all of these modals represent hypotheses. Only may, might, should, will, and would clearly state assumptions in the sentence. The modals can and shall can describe facts and assumptions depending on the circumstances. The modal must expresses necessity. The modal could can either express a suggestion (e.g. "it could do something") or a possibility ("we could show that"). An analysis of 100 randomly chosen sentences with the modal *could* revealed a 40% probability of the sentence representing a suggestion instead of a possibility. Put differently, in 60% of the cases, the modal represented a fact and has therefore been omitted from thelist of hypothetical modal verbs. All in all, we could show that about 5.7% of all sentences contained a clearly stated hypothesis when only accounting for the

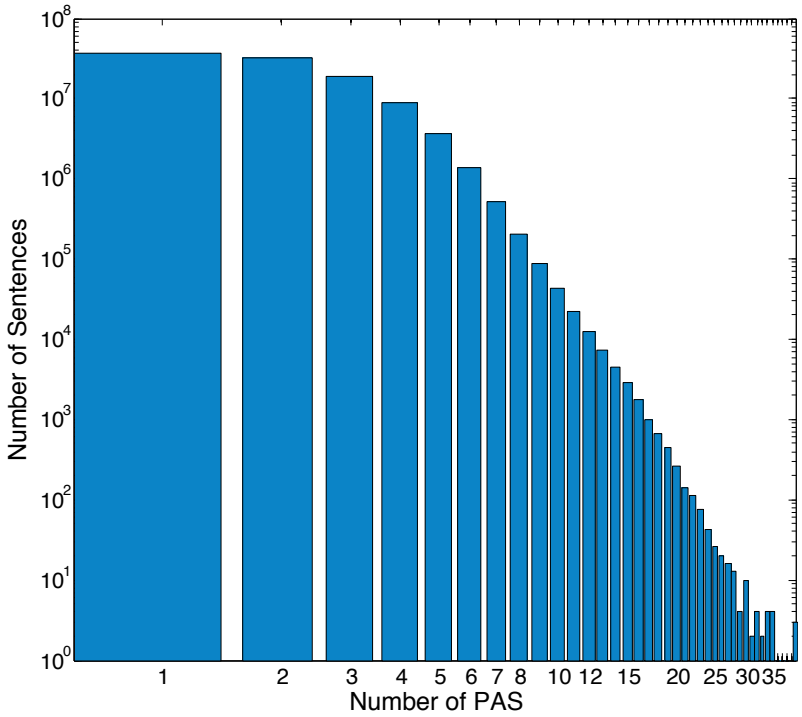
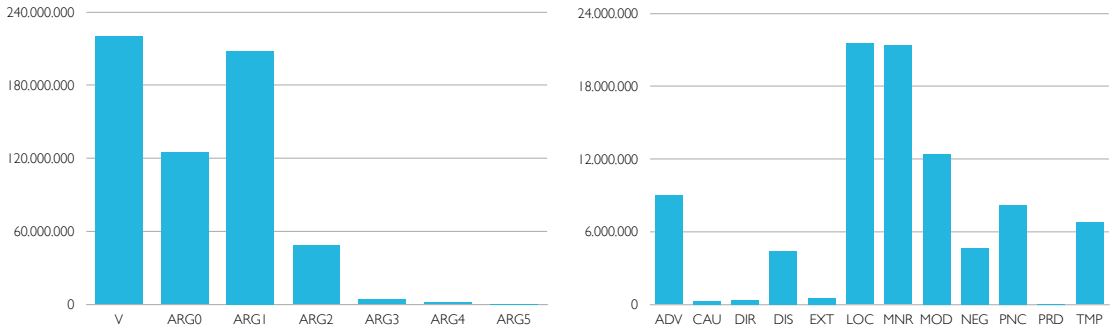


Figure 5.3: Log-log plot of PAS per sentence. Most sentences have between 1 and 3 predicate argument structures. The number of sentences without PAS is not plotted. The log-log plot clearly shows an exponential decrease in the number of sentences for different PAS. Data as at 2011/12/31.



(a) (b)

Figure 5.4: Frequencies of different semantic role labels. (a) shows the absolute number of semantic roles for the main arguments. For comparison, the number of verbs (V) have also been plottet, representing the total number of predicate argument structures. (b) shows the number of semantic roles for the modifier arguments. Note that these occur an magnitude less times in maximum than the main arguments. Most prominent are the location (LOC) and manner (MNR). Data as at Oct 24, 2011. Numbers are for the SENNA SRL package. The width of the bars has no meaning.

previously selected modal verbs and ignoring other phrasings such as “probably” or “the results suggest that” (Hummel, 2011).

Modal	Frequency	Percentage
can	5,399,485	28.7%
may	4,963,417	26.4%
could	2,568,536	13.7%
should	1,963,444	10.5%
will	1,369,375	7.3%
would	1,009,346	5.4%
might	882,664	4.7%
must	615,503	3.3%
shall	11,321	0.1%
other	602,997	3.2%
total	18,783,694	100%

Table 5.1: Distribution of modal verbs. Given are the absolute frequencies of the 9 modals for all sentences in Excerbt’s corpus. The percentages are relative to the total amount of modals. Adapted from Hummel (2011).

Semantic role labeling does, unfortunately, not yet support the hierarchies between PAS. If the main clause states “data suggests that”, then the assertions of the subordinate clause likely represent assumptions instead of hard facts, even if they are not phrased that way. This scenario cannot be detected yet without extensive post-processing but will likely be included in future versions of the ShallowSRL engine (Section 2.5.4).

Another often overlooked category of sentences are negations. While contributing only a little percentage of about 2.8% of all sentences (according to the frequency of the `ARGM_NEG` annotation in PAS data), extracted assertions can completely falsify any result if the negation is not properly handled. SRL outputs negations into the `ARGM_NEG` tag. An analysis of the frequency distribution of terms occurring in this tag showed that in 99.5% of the cases the term “not” is used to describe the negation (Table 5.2).

Negation	Frequency	Percentage
not	6,352,634	99.5%
never	11,713	0.2%
nor	4,196	0.1%
other	16,347	0.2%
total	6,384,890	100%

Table 5.2: Distribution of negations. Given are the absolute frequencies of negations observed in Excerbt’s corpus. Nearly all negations are the single word “not”, whereas “never” and “nor” are negligible in occurrence. Adapted from Hummel (2011).

A deeper analysis of negated sentences has yet to be performed. There is, however, no apparent reason, why they would behave structurally different than sentences without negations. Negations cannot be retrieved via the web interface or RESTful web services at this time. There is, however, effort underway to more closely evaluate the importance of relations. Within the scope of the Negatome Database Project (Smialowski et al., 2010), I am in collaboration with Philipp Blohm from my institute

and Dimitrij Frishman from the Technical University Munich in order to assess the ability of the text mining system to provide an automated method for building the next version of the database of proven non-interacting proteins, the negatome.

5.2 Analysis of Entities and Hits

As outlined in Section 4.3, we have constructed an in-house meta-ontology for entities and entity types. This meta-ontology currently contains about 70 million unique names, of which approximately 580.000 have been found in the text by Excerbt's NER module. This corresponds to 5.8 % of all entities. The relatively low number can be explained by examining the composition of the meta ontology. PubChem is by far the largest resource, comprising 84.2 % of the total number of names. It contains however a large number of IUPAC[71] nomenclature and other chemical names that do not occur in this form in the text. Nearly every time, the colloquial name of a metabolite or drug is used instead of its nomenclature name.

Fig. 5.5 shows the distribution of the number of entities of different types (gene, phenotype, etc.) in the meta-ontology compared to the distribution after named entity recognition as a radar chart. The plot is without the metabolites entity type and shows only the remaining 15.8% other types.

Interestingly, most categories tend to produce much more hits than there are entity names in the ontology. Pathways and cell components are the most prominent examples for this. Other types, such as genes or miRNAs have a high fraction in the number of entities but produce less hits compared to other entity types. Besides metabolites (not shown in the figure), genes have the second highest number of hits, making up for 79.3 of the remaining entities. For most entity types, the observed shape of the radar chart make sense. A dictionary usually contains terms that are each used more than once. For genes and miRNAs the opposite seems to be true. Many of the entities in the dictionary have no occurrence at all. For genes and other types this likely includes synonyms that are never used in text.

To assert the effects of inflection compared to stemming, I quickly analyzed ten randomly chosen sentences for their differences in entity recognition. For these particular sentences, only 3 posed no errors in NER in the stemmed case. On average 1.7 wrong hits were detected for each sentence. Most common errors included ambiguities in differencing between nouns and verbs or adjectives where only nouns can be proper entities. An additional check for the part-of-speech tag could help in these cases. The inflected variant does not change verbs or adjectives to their stem, therefore the noun can be detected exactly as intended from the ontology. To give a short example, in the sentence

> "*NAD* dependent *malate dehydrogenases* of three *trematode* species, *Notocotylus attenuatus*, *N. ephemera* and *N. imbricatus*, have been investigated by *electrophoresis*." (Paulauskas, 1990)

our NER module correctly identified six hits: NAD malate dehydrogenase, notocotylus attenuatus, trematode, emphera, electrophoresis. Imbricatus was not identified because it seemed not to be in our meta-ontology. All possible hits were marked italic. However, it also wrongly detected *malation* and *investigation* as hits due to failed stemming. This example also shows that it is important to allow word

[71] International Union of Pure and Applied Chemistry

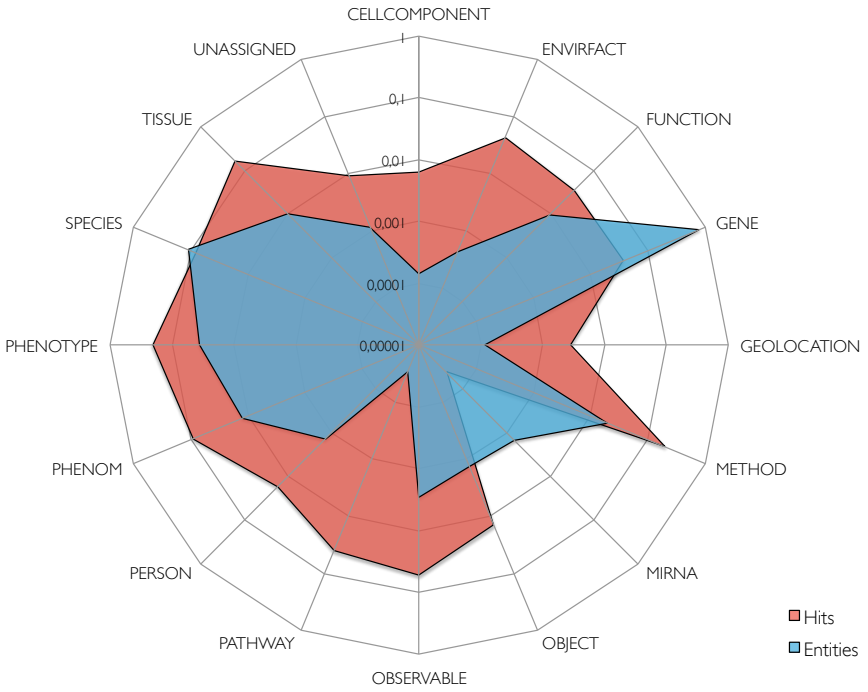


Figure 5.5: Named entity recognition statistics. Logarithmized values of the number of entities and the number of hits these entities have produced in percent. It is interesting to note that while the values for entities (blue) are much more diversified, the number of hits (red) smoothes out these differences. For example, while most of the entities are gene names, the biggest amount of a single type is the methods. Metabolites have been omitted. Data as at 2011-09-27.

insertions for entities longer than two words. If we would not allow an insertion in the entity *NAD malate dehydrogenase*, no hit would have been found in this sentence due to the insertion *dependent* between the first and second word.

5.3 A Graph-theoretical Analysis of Excerbt's Relations

Excerbt's relations basically represent a graph (as they are stored in a topic map). Assuming a graph in the mathematical sense, it consists of roughly 175 million edges[72] and around 580,000 nodes. Any relation in Excerbt is backed by literature evidences. A total of 1.44 billion such evidences have been detected, meaning that each relation has on average 8.2 evidences attached.

I have created a log-log plot of the number of relations versus the number of evidences, which can be seen in Figure 5.6. The plot clearly shows that most of the relations in Excerbt have exactly one evidence and subsequent numbers of evidences for individual relations drop exponentially. However, as some relations experience up to 10.000 individual evidences, the number of 8.2 evidences per relation on average seems plausible. An example for a relation with more than 11.000 evidences is posed between "cells" and "enzymes". One could argue that this is not a real relation as it does not contribute significantly to any insights (it is a rather obvious relation), but restricting those would be against the idea of supporting a multitude of different entity and relation types as we would restrict the output of the knowledge extraction. Another example, with more biological insight is the relation between apoptosis and TNFalpha with 954 evidences throughout literature, spread over all relation types except co-occurrence. This again stresses the fact of stop words, that not only seem to exist in sentences, but also seem to be major contributors to relations, as the various ontologies contain those terms. This also explains the high number of evidences compared to the number of actual PAS. Also within the 1.44 billion evidences, all of them are counted multiple times for each relation, not only distinct ones. At this point, however, we make no distinction between stop-word and non-stop-word relations.

In Figure 5.7, I have analyzed the ratio of all evidences of a particular node in terms of affiliation to specific publications. In other words, the plot shows whether the evidences for relations of a node with its neighbors all belong to different or the same publication. With peaks at around 0.2, 0.5 and 1, the plot shows that, on average, the contribution of publications tends to have peaks for one, two, and five evidences to a node's relations.

In another analysis, I have examined the number of relations per year, as can be seen in Figure 5.8. I have limited the data to gene-gene interactions as those are the ones with the best extraction performance and the least possibilities of false-positives due to exact search in the NER step (see Section 4.3). The number of relations was computed for each year and then normalized by the number of extracted sentences for that year to account for the fact that in later years more articles were published and those publications typically became slightly longer.

The figure basically shows the percentage of relations in sentences. A value of 0.02 means that in that year the chance of a sentence describing a gene-gene relation is at 2 %. This percentage seems

[72] This includes only semantic relations, but no co-occurrences. If we would allow co-occurrences for this analysis, the number would rise to approximately 4 billion edges.

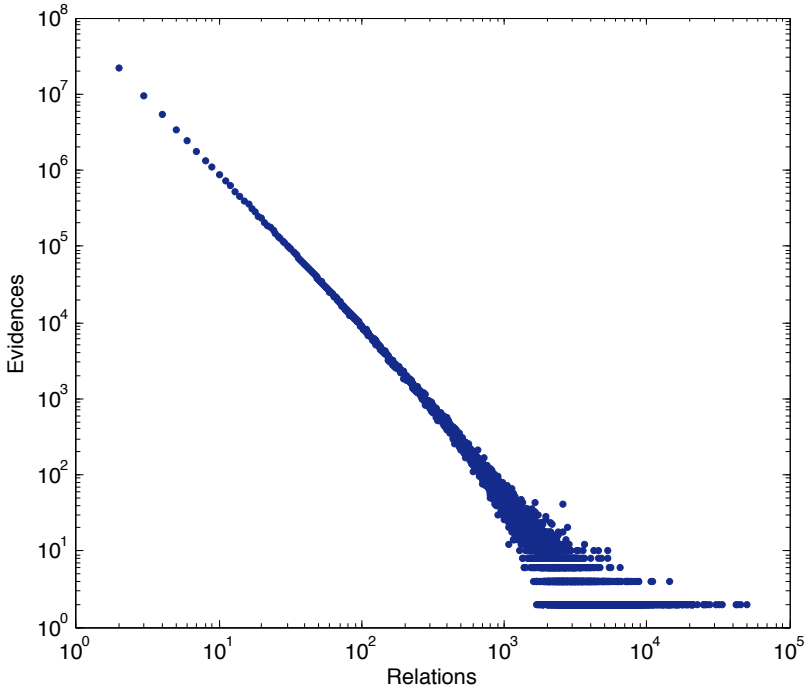


Figure 5.6: Log-log plot of relations vs. evidences. The Figure shows that most relations have little evidences and only few relations have many evidences. Since this is a log-log plot, the descending gradient means an exponential decline. Data as at 2011/12/31.

to drastically increase between the years 1980 and 2000 indicating that either publications report more on genes or that the detection rate of relations increases with publications written after 1990. Another factor, obviously, is that Pubmed citations before 1990 usually or often only consist of a title that contains no verbs and therefore cannot output any relations based on semantic role labeling, which is dependent on the presence of at least one verb.

Interestingly, the absolute peak of 0.027 is at the year 2000 with a consistent, slight decrease afterwards. Whether this is a statistically significant observation or not only the future can tell. However, assuming that this is indeed significant, an explanation for a reduced output of gene-gene interactions in biomedical literature can basically have two reasons: a significantly changed style of writing which makes it more difficult for the semantic role labeling approach to correctly analyze sentences or a general loss of interest in plain gene-gene (or protein-protein) interactions or at least in their publication. As the n-gram trends module in Excerbt's web interface can show, the research area of Systems Biology also started around the years 2000 to 2002 with a drastically increased interest in subsequent years (Figure 5.9). This interest represents a paradigm shift in how biomedical research is performed. The one-gene approach that analyzed every single detail for a single gene became less important and was revised to a more holistic approach of analyzing the overall position of multiple genes involved in particular processes. One could therefore argue that the paradigm shift towards Systems Biology might influence the reporting of scientists on single genes and instead focus more on the involvement in larger processes. However, as said before, if this development continues for the next couple of years,

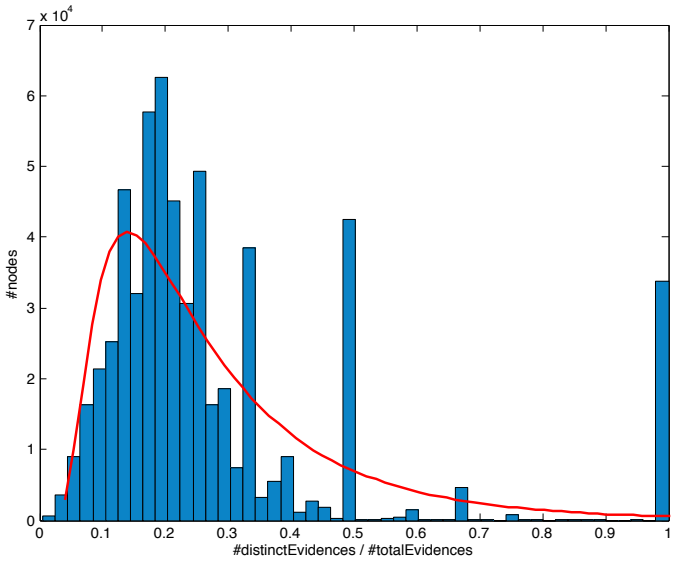


Figure 5.7: Frequency of evidences per publication vs. nodes. Histogram of the ratio of the number of distinct evidences (publications) and the number of total evidences on the x-axis and the number of nodes with that ratio on the y-axis. Colloquially, this gives the number of nodes that have a particular distribution of evidences and their publications. A value of 1 means that the number of publications for a specific node is the same as the number of evidences, meaning each publication contributes exactly one evidence. A value of 0.1 on the other hand means that each publication for that node contributes 10 evidences on average. Basically, the histogram shows the contribution publications make on average to the edges of a node to its size-1 neighborhood. The red line shows a fitted Gaussian probability distribution. The large amount of nodes at the value 0.5 basically means that the neighborhood network contains exactly 2 evidences for each publication. An explanation could be nodes with a degree of 1 and have an activation of inhibition relation type. This relation type is included in the regulation relation type and as such the respective sentence contributes two identical evidences to the different relation types. Data as at 2011/12/31.

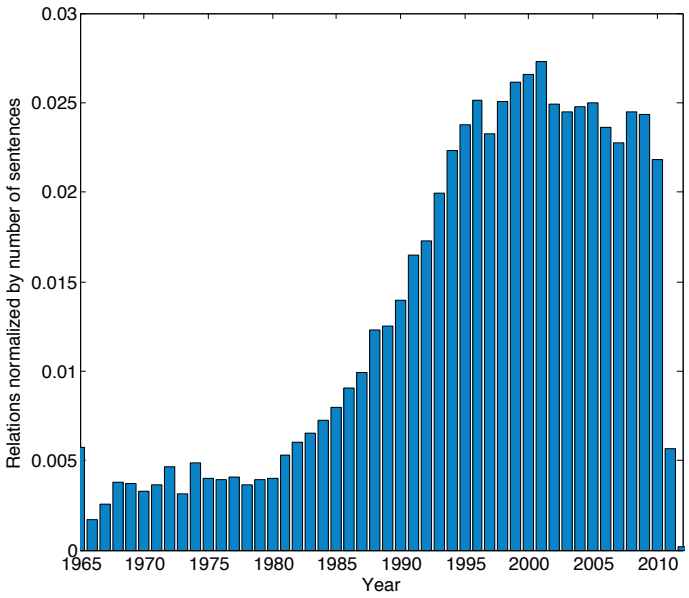


Figure 5.8: Number of gene-gene relations per year. The absolute number of relations for each year was determined and then normalized by the number of sentences published in that year, accounting for the fact that later years produced more and longer articles. Interestingly, the number of extracted gene-gene relations for articles also seems to increase over the years. The values for 2011 and 2012 do not represent valid numbers as the data is incomplete for those timeframes. Data as at July 2011.

this paradigm shift could indeed be responsible. If true, a need to concentrate on other aspects of text mining than gene-gene interactions might be advisable for the future.

Another possible explanation for this behaviour could be the fact that the human genome was published around that time(Lander et al., 2001). Until then, the number of genes having citations increased constantly but afterwards no more genes could be found.

5.3.1 Overall Graph Properties

The main point of studying Excerbt's relations as a whole was to infer properties of the graph that individual entities contribute to. As introduced in Section 2.1.1, networks can be classified according to global graph properties, such as the degree distribution and the clustering coefficient distribution. As for most other biological networks, one would expect a hierarchical scale-free network architecture for the literature graph. As the network is rather large, I have devised a general concept for various MapReduce algorithms that can perform degree and clustering coefficient distributions. While the algorithm for the degree distribution is straight-forward, the clustering coefficient distribution was computed by an adapted version of the parallel breadth-first search (Algorithm 2.3), similar to the one Suri and Vassilvitskii (2011) gave. Their version, however, could not be used unchanged due to differences of the directed graph structure we have. Within the scope of the bachelor's thesis of Tanzeem Haque (Haque, 2012), we implemented and executed the algorithms on the Excerbt graph.

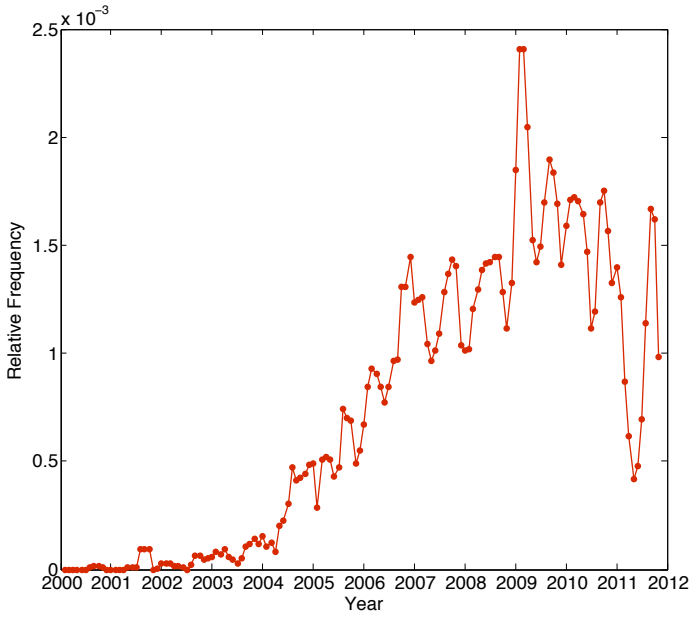


Figure 5.9: NGrams frequencies for the term "systems biology". Data as at 2011/12/31.

The algorithm used to compute clustering coefficients is straight-forward for non-MapReduce applications (Section 2.1). Unfortunately, for large graphs that can only be handled by Big Data approaches, we cannot efficiently determine the number of existing edges within the neighborhood of a node. This would require an ultra-fast lookup function for such types of queries which cannot be implemented because of the nature of MapReduce algorithms and distributed computer architectures. Either the lookup function would have to generate network traffic (which is slow) or would need to have everything within the memory of the server, which is not feasible because of the big data nature of the graph. The idea is therefore to throw a brute-force method onto the problem and generate all paths of length two for any node and afterwards checking all such paths whether they form a triangle. Basically, the clustering coefficient calculation performs three steps:

1. Generate all paths of length two starting from every node in the network. This results in node-triplets *A*–*B*–*C*.
2. Check for each node-triplet that has the same first node *A* whether the third node *C* is a direct neighbor of *A*, i.e. a triplet starting with *A*–*C*– also exists.
3. If yes, count this as a triangle for node *A*, thus increasing the clustering coefficient for node *A*.

An outline of the procedure can be seen in Algorithm 5.1.

Figure 5.10 shows the plotted degree distribution of all 580.000 nodes. Both axes are logarithmized, which shows that the degree distribution follows an exponential decrease for higher degrees. It is consistent with organic networks in that the majority of nodes have a degree of one. Additionally few, very highly connected hub nodes also exist. The shape of the clustering coefficient curve in Figure 5.11 also exhibits an exponential decrease, suggesting a (near) hierarchical scale-free network structure, consistent with previous considerations (Haque, 2012).

```
function REDUCE1(node n, neighbors N):                    create all triangle combinations for each node
1: for each e ∈ N do
2:     emit({n, e}, n))                                   generate a triangle back to source
3: end for
4: C ← GENERATECOMBINATIONS(N)                            generate all tuples of nodes n in N
5: for each c ∈ C do
6:     emit({c1, c2}, n)
7: end for

function REDUCE2(edge e, vertices V):                     check whether these triangles exist
8: if V contains SOURCE(e) then
9:     if |{V \ SOURCE(e) | == 1 then
10:        for each v ∈ V do
11:            emit(v, 1)                                 emit a 1 for each node in V that is not also in e
12:        end for
13:    end if
14: end if
```

Algorithm 5.1: MapReduce version of the clustering coefficient distribution. This version is based upon the NodeIterator algorithm as published in Suri and Vassilvitskii (2011). All steps are reducers, the mappers are omitted because they only bring the input data into the correct format. The first reducer basically creates all possible triplets of two neighbors and the source node A that could exist for a node n: B–C–A. It also includes a triplet back to the source node (e.g. A–B–A) for later use. The trick is now that the first tuple of the triplet is used as the key for the next phase and the third element as the value. With this, the second reducer has for each hypothetical triplet the information whether it describes a triangle or not, namely if and only if the list of vertices includes the source node of the edge in the reducer's key. The second reducer can then simply emit a one for each node in the list of vertices. A third reducer (omitted here for simplicity) is needed in order to count those ones which basically give each node n the number of triangles. The final output is the numerator of Equation 2.2 in Section 2.1.

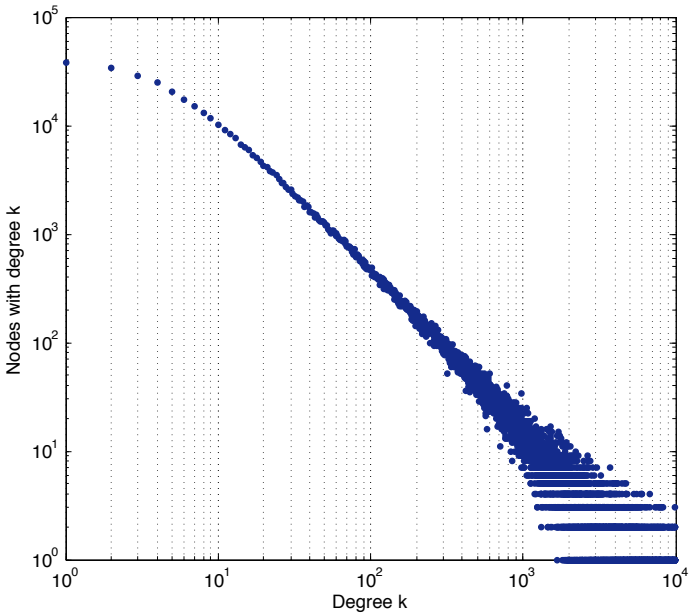


Figure 5.10: Degree distribution of Excerbt's knowledge graph. Frequencies of the degrees k of Excerbt's graph in double-log format. Clearly recognizable is an exponential decrease of the numbers of nodes with a certain degree for higher degrees. The shape is consistent with a scale-free network architecture as described in Section 2.1.1. Data as at 2011/12/31.

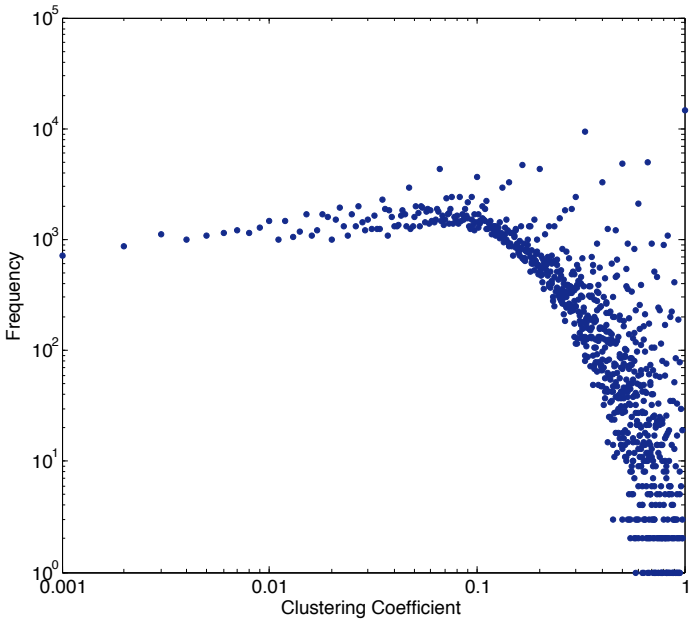


Figure 5.11: Clustering coefficient distribution of Excerbt's knowledge graph. The frequencies of higher clustering coefficients clearly decline towards the score of 1.0. This is consistent with a hierarchical scale-free network architecture as described in Section 2.1.1. Data as at 2011/12/31.

5.4 An N-Gram Analysis

For the Pubmed and Pubmed Central N-Gram Trends corpus as at Dec 31st 2011, I have computed all n-grams with $n = \{1, 2, \ldots, 8\}$, which totaled to about 3.8 billion n-grams in the corpus. In order to be able to derive useful frequency comparison information, the absolute frequencies had to be normalized. In the approach of Michel et al. (2011), the absolute frequency numbers are divided by either the number of documents the words were published in or the number of pages the words were found on. In Excerbt, we relativize the absolute frequencies by dividing them by either the number of documents at that time or the number of sentences to account for the fact that articles tend to become longer over time.

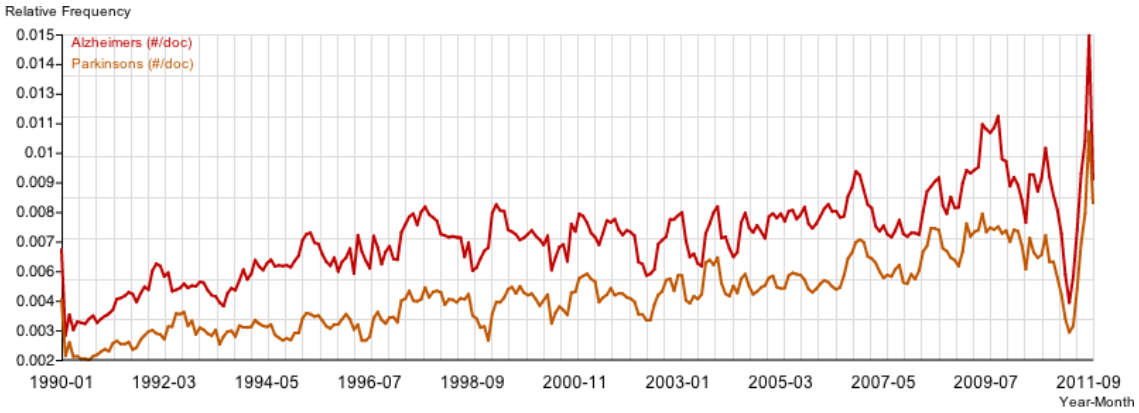
A problem is with data before 1990. When normalizing by the number of documents, a common problem are articles that only consist of a title but no text body. There, a specific n-gram can be only mentioned once per article. Today's articles are usually longer and contain more text, therefore a single n-gram can occur multiple times throughout the document. When normalizing by the number of sentences, the outcome is better. Some sentences within a document, however, might be more important than others, e.g. the title is more important than a sentence within the methods part. A distinction into different importance levels for different sentences can, however, not yet be performed because of limited data.

As the method allows for a relative comparison between any two terms, the normalization bias as mentioned previously is not of importance, as the relative difference between the frequencies of any two terms stays the same as long as they are normalized by the same underlying principle.

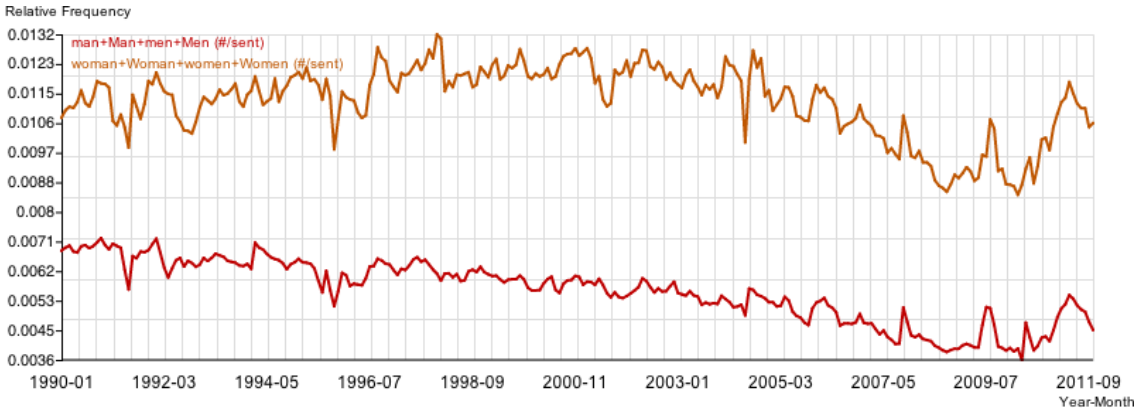
Figure 5.12 shows some examples for the comparison of different words and phrases. In Subfigure 5.12(a), the total occurrences for the disease names *Alzheimer's* and *Parkinson's* are displayed. Generally speaking, the interest in both diseases seems to increase while the relative difference between the two remains the same. Also, interest in those diseases seems to rise and fall at the same time each, meaning that the two diseases are more likely very similar to each other since research on one disease also influences research on the other disease.

Another example is the analysis of the word *man* compared to the word *woman* (in all different notations, see Figure 5.12(b)). Women tend to be mentioned more often throughout the course of biomedical literature. A search for the terms *male* and *female* (not shown in the figures), however, seems to indicate the opposite. This suggests that these differences are mostly due to the writing style of authors who prefer the word male over the word female, but prefer the word women over the word men.

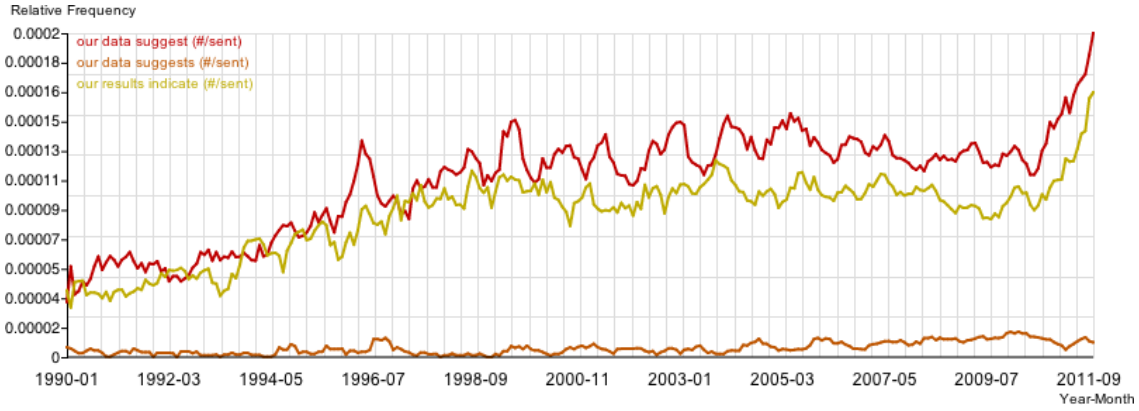
A last example in Figure 5.12(c) analyzes the usage of three different phrases *our data suggest*, *our data suggests*, and *our data indicate*. While the first and last phrase are relatively consistently used throughout biomedical literature, the grammatically wrong phrase connecting *data* (a plurale tantum) with the singular form of the verb *suggest* is used much less frequently but has occurrences despite the fact that correct grammar and spelling should be mandatory for published articles.



(a) Alzheimers vs. Parkinsons disease



(b) Man vs. woman



(c) Suggest vs. indicate

Figure 5.12: N-Gram trends examples. (a) Frequencies of the diseases Alzheimer's and Parkinson's. The spike at around 2011 is an artifact because of limited data. (b) Mentions of man versus the mentions of woman. The word *woman* seems to be of much higher frequency everytime. (c) An example that compares the phrasing of two different subordinate clause introductions. Note that the grammatically wrong version *our data suggests* (with a third person s) is still used throughout publications but at much lower frequency. All data as at 2011/12/31.

5.4.1 The Size of the Biomedical Domain Language

Excerbt's N-Gram Trends corpus consists of roughly 8.3 million distinct words (n-grams with $n = 1$, unigrams) when taking different writings of the same word, for example with capital and small letters (e.g. word, Word) into account and omitting words that contain special symbols[73]. Michel et al. (2011) reported that they counted 544.000 distinct words in the year 1900 and 1.022.000 distinct words for the year 2000 for the English language in the studies they performed for their Google Books project. Assuming a linear increase in the number of new words throughout the years, an extrapolation of the numbers by a linear function[74] gives a rough estimate of 1.074.600 distinct English words for the year 2011. While the corpus definitely would contain more than these 1 million distinct terms, they focused their analyses onto *common* words, which is defined as an occurrence of at least once per billion total words in the corpus. Also, they created a heuristic that tracks "wrong" English words, i.e. words that contain numbers, are only numbers, or are foreign words. They estimated that of all total words, 51% fell into these categories in 1900 and 31% in the year 2000. Another linear fitting gives an estimate of 28.8% of all words to fall into one of the non-word categories for the year 2011.

I used the same measures for commonness on the Excerbt N-Gram corpus. The total number of non-unique unigrams of Excerbt is at 2.5 billion, which is the total number of non-distinct words in the corpus. An occurrence of once per billion leaves us with an absolute frequency of at least 2.53 occurrences in total. Counting each unique unigram consisting only of word characters with 2.5 billion words in total gives us exactly 7.068.080 common words in the Excerbt corpus, according to the definition by Michel et al. (2011). Subtracting from that the extrapolated number of common English words for the year 2011 and subtracting the percentage of non-word unigrams gives an estimate of the number of terms specific to the biomedical domain. With this approach I identified roughly 3.96 million unique biomedical words that do not occur in regular text books.

At a first glance, this number seems unrealistically high. However, frequent words in biomedicine are gene names, disease names, and metabolites that all can have numbers as constituents of words, not forgetting abbreviations and others. The above correction for these non-words based on regular texts makes this number somewhat difficult to compare with the exact number of new domain language words. Excerbt contains about 600.000 distinct entity names that have been de facto found as hits for relations. Subtracting that number again from the 3.96 million gives an estimate of 3.36 million unique English biomedical words that do not occur in our meta-ontology but can be assumed of relevance to the biomedical domain because of the above commonness definition. This number is still relatively high which leads to the idea that a dictionary-based named entity recognition includes a high number of used terms but still lacks a big part of all biomedical terms. Increasing the number of ontologies for Excerbt's meta-ontology in order to increase the number of covered terms would, however, only marginally improve the situation. Our meta-ontology contains the most common ontologies with the most entries, therefore any other ontology can only introduce a minor number of new words to the corpus. In spite of that, the complete biomedical domain language includes more than just entity names which are all nouns. Additionally, verbs or adjectives are common and make up a big part of the domain language.

[73] In various programming languages, such as Java, word character symbols are defined as letters, numbers, and dashes (a-z, A-Z, 0-9, -, _). Everything else is considered a non-word character. This definition was used here.

[74] $f(x) = ax + b$, fitted to $a = 4780$, $b = -8.538.000$

The most interesting fact, however, is that apparently the biomedical domain language seems to be larger than the usual English language by a factor of three, even under the assumption that the method is only an approximation.

5.5 Evaluation

The usual approach to evaluating any technical invention usually is to compare the results of the system to some gold standard. While some systems, such as simpler heuristic methods tend to perform well in such an evaluation, more complex systems such as text mining applications usually perform poorly because of unclear definitions of the gold standard. The main reason for that is that a new TM system usually implements a completely new approach for which no gold standard has been defined yet.

Consider an example gold standard that was developed for evaluating TM systems that recognizes gene-gene interactions. TM systems that have been developed to perform this task can be evaluated with good confidence against that gold standard. However, systems such as Excerbt that have a completely different spirit and purpose[75] will very likely perform poorly against that evaluation. One could, as a matter of course, only evaluate overlapping parts of the systems, but ultimately this does not allow for a exhaustive comparison.

It should be added that, while evaluation is important, not too much emphasis should be put onto the results of such evaluations. Compared to specific algorithms that try to predict certain features, such as protein structures or miRNA target sites, and that can be successfully assessed, TM behaves differently. The main reason is that in TM, we do not compare the performance of a system against something that can be experimentally verified, but against a human who is unable to extract factual assertions consistently and objectively from all available literature. Text extraction cannot be done through experiments. Put differently, a scientist that reads articles always extracts a subjective interpretation of some of the actual knowledge put on paper instead of a reproducible and objective summary. Comparing against a subjective interpretation therefore seems superfluous.

Because of this, the correct question to ask when evaluating text mining systems is not "how does it compare to an arbitrarily defined test set?". Instead, a scientist should rather ask "how can this system help me achieve my goals and does it deliver enough of the right information?". The main problem always is that a scientist cannot read all available literature on a certain topic, but a computer can. Even if the computer does not (yet) understand everything that is written to an extent a person could with enough time on hands, it can provide an enormous help to any user.

In my opinion, Hersh (2005) gets exactly to the point of the problem in the conclusion of his article:

> "[...] begin to focus on user-oriented evaluations to determine how systems will be most effective in real-world settings."

His focus on user-oriented evaluations is however not to be confused with an evaluation of the frontend design of an application. While of importance in the evaluation (a user should be able to perform efficiently with a system), the main point is to concentrate on features that are able to gain new

[75] As a reminder: We want to be able to extract a multitude of different types of facts from text with a common approach.

insights that are useful to the user, which in turn reflects back on the actual backend features and implementations.

Despite this, I will attempt a partial, so-called *data-driven* evaluation of the Excerbt system against the performance of other, similar systems and gold standards that have been developed over the last couple of years. Additionally I will also summarize some operations that attempted an evaluation in the above described more *user-driven* sense.

5.5.1 Data-Driven Evaluations

BioNLP 2011

In order to perform a data-driven evaluation of the contents of the text mining system Excerbt, I have evaluated the relations against two other systems: the BioNLP shared task 2011 for entity-relation recognition [76] and the BioInfer corpus (Pyysalo et al., 2007).

The BioNLP shared task 2011 for entity-relation recognition consists of training and development data where 470 Pubmed abstracts are annotated with relations. These relations are either protein-protein binding or sub-complex forming. In order to evaluate Excerbt, I have used this data and transformed it in a way that it is useable for an informative evaluation. Since an exact mapping between their relation types and our relation types was not possible, I have omitted the relation type information and just performed an analysis whether the involved entities have been found in any semantic relationship. In order to not taint the pool of data that is tested, I did not include co-occurrence information from Excerbt. I then used only the given Pubmed abstracts and extracted all relations of these documents that were found by my system. Additionally, I omitted all relations between entities that do not occur in our corpus. Initially, the corpus contained 1890 relations. After these limitations, 49 relations remained. This extremely low number compared to before demonstrates one significant point: an evaluation can only be as good as the data that it is compared to. The BioNLP corpus obviously contains many annotations that do not directly exist in gene/protein ontologies, such as Entrez Gene or Interpro. Manual inspection of the entities that did not exist in our corpus seem to signify a pattern: Often, entities such as “CD3/CD28” which clearly denote an alternative between two entities instead of a single one, or terms such as “-52 and +89 nucleotides” which rather seem to be positional information on DNA strands, are annotated in the BioNLP corpus. Due to our architecture, such types of entities cannot reliably be determined and have therefore been omitted in Excerbt.

After comparison of the existence of these remaining 49 relations within Excerbt’s corpus, only six common relations could be identified. This low number seems to attest a very bad performance. However, together with the previously mentioned fact of annotated entities in their corpus that do not exist in our included ontologies, and a manual random inspection of the not-found relations one thing becomes clear. Sentences are in many cases wrongly annotated. For example, the relation between “IL-2” and the entity “sites” does not exist in this sentence:

> However, CsA and FK506 inhibit the appearance of DNA binding activity of factors that bind to the NF-AT and AP-1 sites in the IL-2 enhancer (Granelli-Piperno, 1992).

[76] http://2011.bionlp-st.org/, accessed 2012-08

Clearly, this states that while the actual evaluation performs very poorly, the result is not significant at all and therefore cannot be used in order to determine the performance of the Excerbt text mining system. With this type of evaluation, the problem always seems to be with the original data the system is compared against. If this data contains errors that are not simply data glitches (e.g. less than 1 per 1000 annotations is wrong) then the determination of neither false positives nor true positives are a significant indication for a fully functioning system. Also, because of the radical changes that had to be made to the underlying corpus in order to fit to a data model that Excerbt can be evaluated against, I did not perform a comparison of my results to the official participants of the BioNLP 2011 shared task whose systems where built in order to comply to the test.

BioInfer

In collaboration with Robert Strache (2012), one of my master's students, we performed a similar analysis on the BioInfer corpus. For this work, the important evaluative measure was to determine whether the improvements to the semantic role labeling step by limiting long arguments to the first occurrence of a new verb were significant (see Section 4.4). Results indicated that the F-measure increases slightly.

As a conclusion, the results of this evaluation were similar to the BioNLP evaluation. Additionally, our approach produced a very high number of seemingly false positives (according to the BioInfer corpus) that were, in fact, not all wrong. Manual random inspection of the BioInfer corpus showed that many of the assertions in the text are simply not annotated but found by our approach. Overall conclusion of the BioInfer evaluation was that the corpus is not suited to evaluate a general-purpose text mining system such as Excerbt, mostly because the corpus contained too few annotated relations (Strache, 2012).

This shows again, that user-driven evaluations are necessary and better suited at determining the performance of a text mining system.

5.5.2 SIDER Evaluation

Together with Nikolas Gross (2011), also a master's student under my supervision, we evaluated our text mining system in the context of relations between genes, drugs, and side effects. For this, we introduced another ontology into the system, the SIDER (SIDe Effect Resources) database by Kuhn et al. (2010). The SIDER database contains a list of side effect and related drugs that can cause these side effects. These relations were extracted from package inserts of drugs approved by the FDA (the United States Food and Drug Administration).

In this work, we extracted all drug—side effect relations from SIDER and compared them with all drug—side effect relations that were found in Excerbt. After careful evaluation, we identified that approximately 88% of the entities in SIDER could be mapped to entities in the Excerbt corpus. An examination of the co-occurrence information between those drug and side effect entities showed that only 32% of the interactions in SIDER could be found co-occurring in sentences in Excerbt. A probable explanation for this is that interactions between drugs and the side effects they cause are usually not

explicitly mentioned in biomedical research publications but probably rather medical trials and similar studies (Gross, 2011). It is important to note that this low number stems from the fact that the two databases, SIDER and Excerbt, are constructed from two very different sources, namely drug package inserts and biomedical publications.

Setting these 32% of all relations as a new baseline, we identified 50% of those co-occurrences to be in semantic relation to each other as found in Excerbt. When bearing in mind that co-occurrences within a sentence only produce relations in between 2 and 5 % of all cases[77], the above result of 50% relations found seems highly significant.

This evaluation is still data-driven, but is not against a general purpose corpus but a very specialized one. The results suggest that we find 50% of what we could theoretically find. Compared to a human who can read and understand very little amounts of literature in any given time, this is a huge gain. It is to note that it is unclear whether the 50% is everything that can be found or if the number would increase by error reduction or method improvement.

5.6 Applications and Use Cases

In the last section of this chapter, I want to highlight some applications that have been performed with the techniques introduced in previous parts. Also, I want to highlight, how the created application stack can be used as a systems-biological modeling suite in order to gain insights into specific biomedical processes. Whenever possible, the queries performed to Excerbt (specifically to the web interface or the query web service) will be given.

As a side note: these examples can also be understood as methods of a user-driven evaluation of the system and show that despite a relatively moderate performance of data-driven evaluation approaches, systems can be successfully used in relatively short time to build qualitative knowledge models. Main measure in user-driven evaluations should be amount of energy (mostly time) that is put to work with such systems.

5.6.1 Compiling an Overview and Answering Specific Questions

The Excerbt suite allows a multitude of different workflows for gaining knowledge. The most basic usage scenario is to compile an overview of a particular entity, for example a disease such as Parkinson's disease (PD). Parkinson's disease is a nervous system disease affecting dopaminergic cells in the brain. While this is not an introduction to the disease, the most common symptoms of PD patients are outlined in Figure 5.13. It is the second most prevalent disease in the elderly population (Forman et al., 2004), making this an enormously important research target.

Now, in order to gain a better understanding of the disease's pathogenesis, a first step in Excerbt would be to search for neighbors of the entity "Parkinson's disease". In biomedicine often the genetic causes for the disease are most interesting. A second step would therefore be to concentrate on

[77] When considering a total of 150 million relations and 4 billion co-occurrences as was the case in Wachinger and Stümpflen (2010), only in 3.75% of all cases a co-occurrence is also a relation.

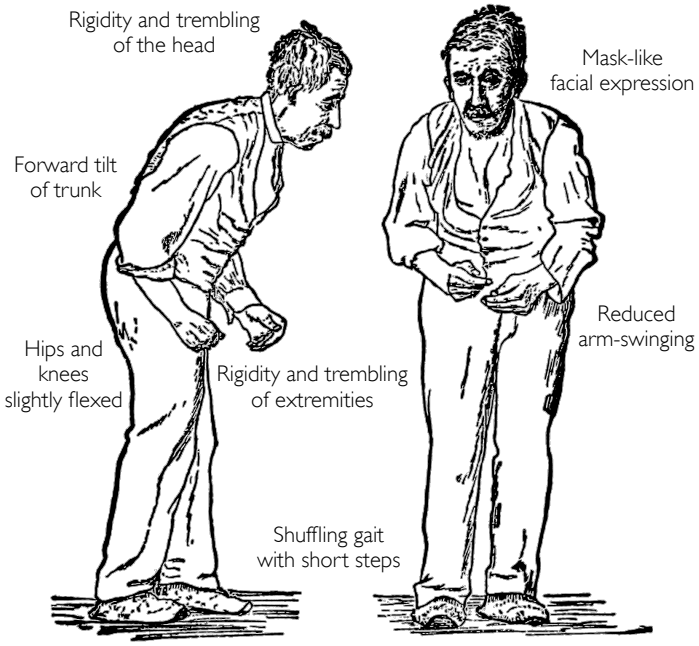


Figure 5.13: Parkinson's disease (PD). Patients with PD usually suffer from multiple symptoms. Most prevalent are a the typical trembling of extremities and the head, a tilted forward trunk and a short-stepped shuffling gait. Figure extended with typical symptom descriptions and adapted from Gowers (1892).

gene/protein neighbors. This type of query can, for example, be performed with the query `PHENOTYPE "Parkinson's disease" ANY_INTERACTION GENE` from the web interface.

With this knowledge, one can proceed further. A next step would be to find all connections between the neighbors of PD in order to get a better understanding of their interconnectedness. This could be performed by a query such as `GENE gene1 gene2 gene3 ... ANY_INTERACTION GENE gene1 gene2 gene3 ...`. Finally, a computer cannot automatically perform everything. Therefore the scientist has to manually inspect and filter the network for wrong and uninteresting areas. The process is outlined in Figure 5.14.

Figure 5.15 shows the application of this process to Parkinson's disease. In the end is a network that contains the most important players in the disease. Comparing this to a Cell Snapshot (Shin et al., 2009) of the same process exposes a very high similarity of the two networks. Overall, the creation for such a network takes approximately one hour. The network can be created nearly automatically by taking only edges into account with a certain number of evidences. This approach is extremely useful for the creation of figures and knowledge networks that are used for reviews.

Another, relatively straight-forward approach is to answer specific questions. For example: Which genes are responsible for the neurodegenerative damage in the hippocampus in Alzheimer patients? This query could be performed in a multi-step process:

1. Retrieve all gene neighbors of Alzheimer's disease:
   `PHENOTYPE "Alzheimer's disease" ANY_INTERACTION GENE`

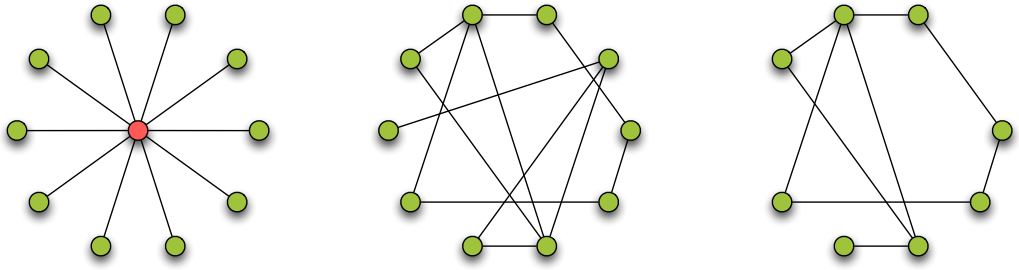
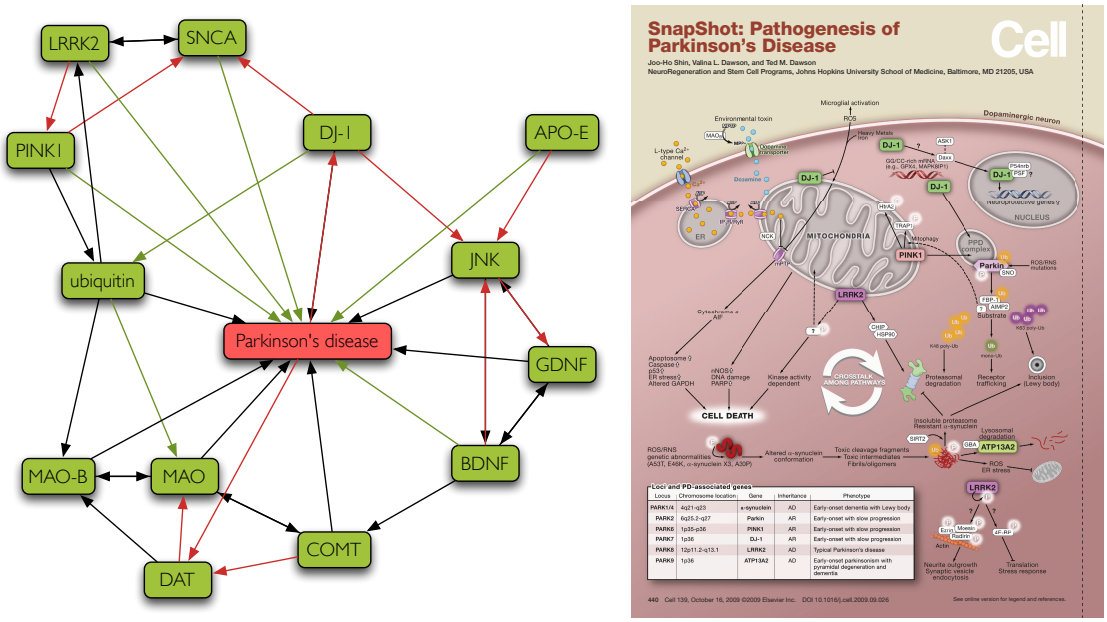


Figure 5.14: Getting an overview – outline. A series of steps towards an overview of a particular process most often begins with some kind of targeting where interesting partners are identified. In the second step, expansion, necessary additional relations are included. Lastly, the network is filtered according to some criteria resulting in a final model.



(a) (b)

Figure 5.15: A snapshot of Parkinson's disease. (a) The network was created with Excerbt. Starting from the disease entity "Parkinson's disease", a series of subsequent query and filtering steps led to the development of this graph. Figure based on work by Michael Greeff. (b) On the right is the Cell Snapshot of PD's pathogenesis (Shin et al., 2009). Closer inspection identified that both models had the same major players and their interactions. This proves that the Excerbt text mining system can be used in order to gain useful insights into the pathogenesis of particular diseases.

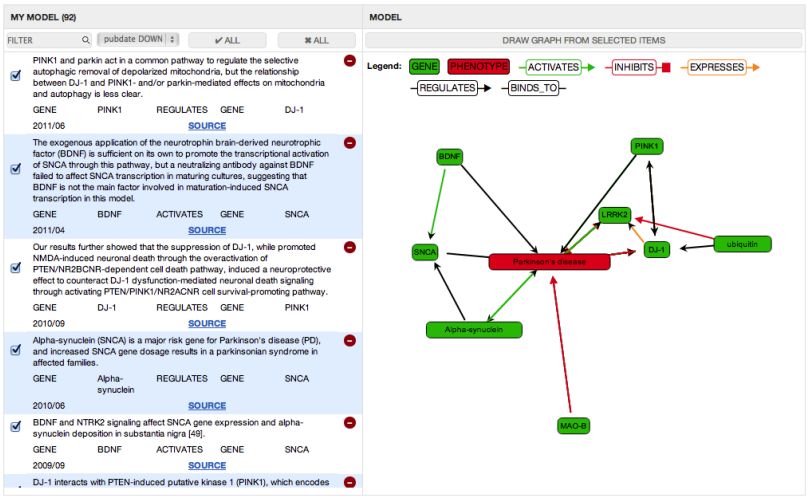


Figure 5.16: Screenshot of the creation of a PD snapshot. The web interface supports queries with multiple sources and targets. Evidences can be added to the model all at once, resulting in a graph similar to the displayed.

2. Retrieve all genes that are expressed in the hippocampus:
   `TISSUE hippocampus EXPRESSES GENE`

3. Build the overlap between the two gene sets and expand/filter the resulting subnetwork

Generating the overlap can currently not be performed via the web interface. However, the interface allows the export of relations in a model in a computer-readable format that can then be used in order to process it locally. This short example shows that it is relatively simple to interactively go from a specific question to a general answer.

As a third example, one could ask the question: What are the genetic factors involved in the effects of the drug Imatinib on Diabetes? This kind of query can easily be answered by the indirect query module in excerbt:

```
METABOLITE Imatinib INDIRECT PHENOTYPE Diabetes VIA GENE
```

This query results in a list of 115 gene names, 13 of which I have randomly included in Figure 5.17. Depending on what is exactly asked, one could, for example search for orthodromic regulatory edges going from Imatinib to Diabetes that would explain an activation of specific genes/proteins by the drug that have a known influence in the disease Diabetes. Models like this can be easily created through the web interface and gives a researcher a tool with which he/she can assess the situation in literature.

5.6.2 Hematopoietic Stem Cells

A second application with the text mining system Excerbt was to support research on hematopoietic stem cells (HSCs). In collaboration with a PhD student colleague from the Institute of Stem Cell Research at the Helmholtz Zentrum München, Konstantinos Kokkaliaris, we used the TM system as a first step in his research. The basic idea is to gain knowledge of the molecular control of HSCs by their micro-

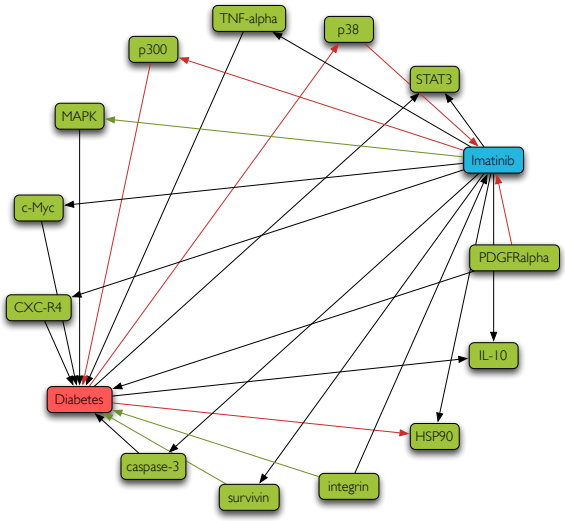


Figure 5.17: How does Imatinib influence Diabetes? The figure shows genes (green) that have edges to both, Imatinib (blue) and Diabetes (red) that could explain the effects of the drug on the disease.

environmental niche, the bone-marrow (Wilson and Trumpp, 2006). Figure 5.18 gives an overview of the differentiation process of HSCs.

For this, we had a list of approximately 300 genes that are preferentially expressed in stroma cells, but can maintain hematopoietic stem cells. The idea was now to check the affiliation of those genes to certain cell fates, such as apoptosis, proliferation, differentiation, adhesion, growth, migration, and others. Since sentences in scientific literature in general and Excerbt in particular do not contain many relations of the form “gene A activates differentiation”, we used an approach of gene-gene interaction annotated with a certain context. In practice this means that we extracted all gene-gene interactions of the 300 genes in the set and annotated them with the corresponding cell fate depending on whether one or both of the two players had a co-occurrence relation with one of the cell fates. The result is now a list of those genes and an enrichment to a certain cell fate depending on the interaction with the other gene. The next step is now to manually check significant hits in cell lines with quantitative real-time PCR (polymerase chain reaction).

5.6.3 Ovarian Cancer Model

In collaboration with Arash Rafii Tabrizi from Cornell University, Qatar, we used the Excerbt text mining system to build a genetic model of ovarian cancer (OC). According to Arash, OC is not a very prevalent cancer, but one of the most lethal for women and therefore of increased interest for his group.

The approach we implemented here was similar to the one with HSC differentiation. Also, one major field of interest was the progression of stem cells according to certain criteria, such as chemoresistance, metastasis, invasion, or epithelial to mesenchymal transition. In a similar approach as above, we

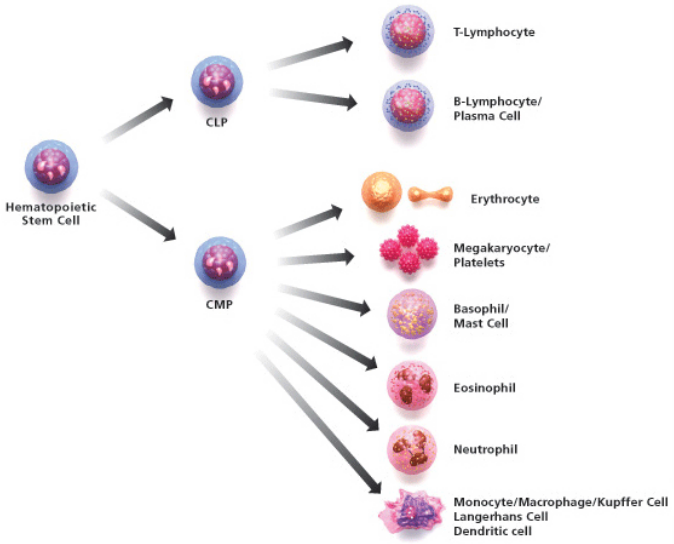


Figure 5.18: Hematopoietic stem cell (HSC) differentiation. Adopted from (Sigma-Aldrich).

conducted a study with certain genes, such as Notch, Interleukines and Jak and determined the context of each interaction by inclusion of the respective criterion.

Figure 5.19 shows the the role of the JAK pathway in ovarian cancer. To build the model, we first conducted a study of all genes involved in ovarian cancer that were positively enriched with one of the criteria. In a second step we searched for all genes related to the JAK pathway by searching for neighbors of the JAK genes. Afterwards, the overlap between the ovarian cancer genes and the JAK pathway genes was determined in visualized as a graph. Evidences were manually corrected for relevance. The remaining network now describes all the genes that are involved in ovarian cancer and are part of the JAK pathway. Further analysis of this network is currently done by Arash and his group.

5.7 Summary

In this chapter, I have examined Excerbt's data corpus, i.e. the sentences, predicate-argument structures, relations etc. that were generated by the methods discussed in the previous chapters. A clear emphasis was on the knowledge graph that contains the relations between various biomedical entities. I have introduced various algorithms and their results that can analyze the knowledge graph in a large-scale fashion, such as degree and clustering coefficient distributions. Additionally I gave an overview of what is possible with regular statistical methods, such as an examination of N-Gram frequencies.

I then presented two methods of evaluation of the text mining system. In one, Excerbt's data was directly compared to training sets originally developed for other text mining systems. I also presented a more user-oriented evaluation approach that shows the possibilities that arise when using the system for specific tasks.

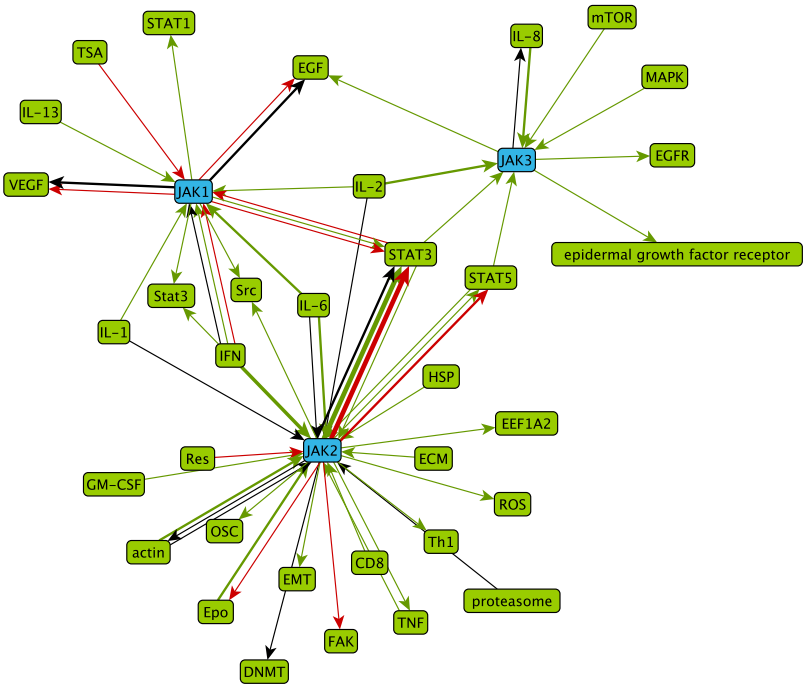


Figure 5.19: Network of genes involved in JAK pathway and in ovarian cancer. Genes were selected by overlap between the two individual networks and an enrichment according to the term criteria chemoresistance, metastasis, invasion, etc.

Finally, I introduced two use cases in collaboration with biologists, where using the text mining system was a first step of research. An in-depth analysis of these first results and a subsequent experimental analysis will be the next steps but should provide useful insights into the respective research areas.

All in all, text mining can tackle many different questions. Especially through the ability to work with big data sets, such as huge literature graphs, results can be discovered faster and with less effort. In the next chapter, I will present another approach to gaining insights from large-scale network data with big data approaches.

6 From Large-Scale to Fine-Grained: Finding Neighborhoods of Interest in Networks

> "An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs."
>
> *Frank E. Grubbs (1969)*

Next generation knowledge extraction is not only about *how* to extract knowledge from unstructured text but also about what to do with it afterwards. While a manual inspection of a knowledge graph, such as the literature network exported by Excerbt and the various applications concluded from this data (Chapter 5) are useful methods when attempting to analyze and advance discovery, automated methods make life easier by providing enormous gains in terms of coverage and selection of information from any resource.

Whenever a scientist is confronted with a problem in his or her research, a common way out is to develop hypotheses or suggestions that could explain the observed data (the problem) and then generating ideas on how to prove them. Translated to biology, this could for example be a biologist that has a "problem" related to a specific disease, which could simply be the desire to understand its pathogenesis. A suggestion in this context would then be a hypothetical novel interaction between two proteins that are involved in this pathogenesis.

In any automatic hypothesis generating system, where from a specific set of input data new insights might be gained from, the primary goal is to achieve the identification of statements that are subject to being helpful. As the work in this thesis is based on a network of entities and their relations extracted from literature, a hypothesis in this context is the proposal for a relation that has not yet been indicated by literature. For this work, I will define a measurable property, called "interestingness" and use that in order to find areas in the literature graph that are objectively interesting to a scientist based on the knowledge stored in the graph.

The basic idea behind this approach is to make the creation for the hypothesis space automatable and more objective for the scientist. While, obviously, completely automatic approaches that not only produces hypotheses or suggestions but rather a list of instructions on what to examine next, is not possible for the time being and not desired because that would require truly intelligent behavior of a computer machine. I think, however, that discovering essential and interesting facts from the current state of affairs is a vital task in order to encourage the scientist to get new ideas on where to proceed with his or her studies.

This chapter aims at providing a method of automated hypothesis generation. Section 6.1 will introduce the concept of interestingness as used in this thesis. Section 6.2 will provide some insights into related

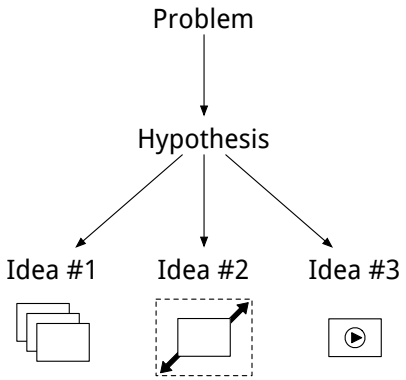


Figure 6.1: Problem, hypothesis, idea. Hypothesis for problem solving should be abundant, objective, and reasonable. Generating hypotheses in an automated way guarantees those properties. Adapted from Holst (2011)

work in the field of automated hypothesis generation. Section 6.3 then presents an idea on how to implement such an approach for the biomedical domain and Section 6.4 will explore big data approaches. Section 6.5 will finally use the methods and approaches from before in order to create an application capable of handling the presented approach.

6.1 What Defines Interest?

There are many definitions of what can be considered *of interest*. Most of them are subjective to the person and depend highly on their personal preferences and the observations they made during their studies and their education. Here, I try to find an objective definition that is on the one hand useful to scientists regardless of their personal preferences and on the other hand applicable to network analysis because the data in Excerbt is a graph.

In the scope of network analysis and the extent of this thesis, I define interestingness as one of:

1. The absence of something that would be expected. Conveyed to the Excerbt graph, this would be either a missing link between two nodes or a missing node with the restriction that these edges are not simply absent but should actually exist (see Figure 6.2(a)). Put differently, this would correspond to an under-representation of the true knowledge within the form of the graph.

2. The presence of something that does seems unexpected. For example, a link between nodes that do not seem to have a true interaction with each other or nodes within network trusses that differ significantly from their surrounding nodes (see Figure 6.2(b)). Put differently, this would seem as an over-representation albeit in the sense that this over-representation might be wrong.

3. A change of direction or the reversal of a statement over the evolutionary development of the graph. An example might be an edge with a positive influence the becomes an edge with a negative influence over time (see Figure 6.2(c)).

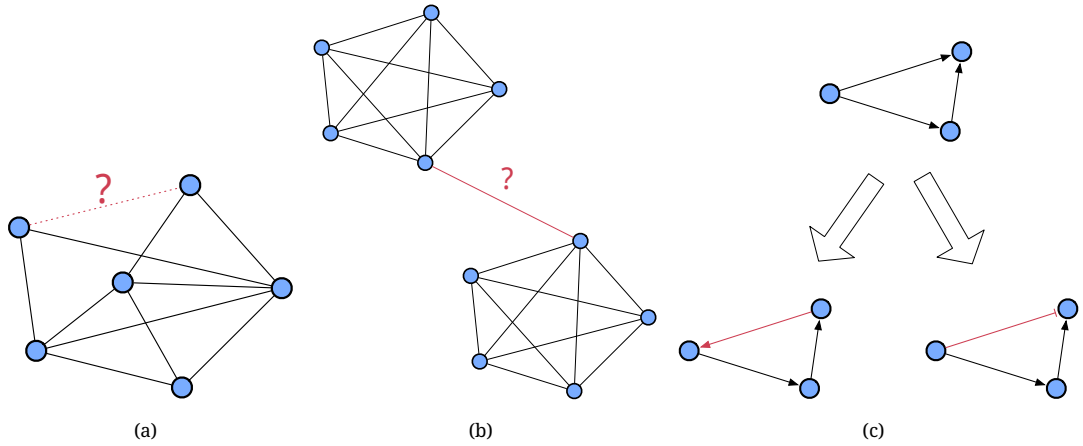


Figure 6.2: The three definitions for interestingness. (a) The red dotted edge might be missing because of the community structure of the surrounding nodes.[78](b) The red line is interesting because it connects two different communities. (c) An edge (top) might evolve over time. The direction or the type of the interactions might change over time.
Dotted edges do not exist in the original graph and are hypothetical, solid edges do exist.

While problems one and two can be computed by only taking the graph's structure into account, the third one needs an additional evolutionary dimension to nodes and edges, the time. Documents in Excerbt have a time stamp assigned which give some kind of time dimension to the approach. Some work has been proposed in the area of dynamic belief changes (Rzhetsky et al., 2009). I will, however, only focus on the other two.

An additional caveat is also the detection of missing nodes for hypothesis generation. Missing nodes cannot be predicted from the graph's structure, since there is no information in the graph whatsoever on what nodes might be missing[79]. Therefore I will concentrate only on missing links that – as will be described later in this chapter – can be detected.

Just to mention it here, the detection of links that are present but unexpected in problem two is similar to the finding of bridges between communities (Figure 6.2(b)). In this case, we would be interested in edges whose deletion from the full network would lead to a collapse of the network into separate subnetworks. However, Cohen (2009) reports that until now no efficient algorithms have been found that can tackle this problem with MapReduce. We will later see that this can be overcome by restricting this to edges within communities that do not fulfill bridge functions (Section 6.4.2).

[78]Note that this is only an example and the graph was drawn to reach that conclusion. Any other edge in this non-complete clique might also be "missing". We will later see, that the probability of edges that are more likely missing than others is quantifiable, see Section 6.4.
[79]This would equal to a prediction of, e.g., genes or diseases, that have not been discovered yet.

6.2 Related Work

Attempting to detect insights in network data has always been of great importance. Areas of interest in networks have been studied extensively in the literature with the most prominent example being an algorithm to detect node importance, called the *PageRank*. In this section I will briefly introduce the notion of node importance in networks and put it into context with the previously definted notion of interest.

PageRank algorithm developed by the founders of Google, Larry Page and Sergey Brin, in 1998 (Page et al., 1998). The basic aim of the algorithm is to compute the relative importance of nodes in a graph, given their surroundings, through an iterative process. In Google's early days this was used as their main method to rank web page results of their search engine and present them to the user in order of relevance.

Simplified, the PageRank gives a likelihood that a user who is randomly clicking on links in web pages will arrive on a particular page. Mathematically, it is defined by the following equation:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)} \tag{6.1}$$

where $p_1, p_2, \ldots$ are the nodes in the graph, $M(p_i)$ is the set of nodes that have an outgoing link to node p_i, $L(p_j)$ is the number of outgoing edges of node p_j, N is the total number of nodes in the graph and d is the so-called damping factor. The damping factor accounts for the fact that a user randomly surfing the web usually stops after a certain amount of clicks and returns to something else. Basically it is a decay factor limiting the amounts of clicks.

The PageRank algorithm, however, is not able to make a contribution to finding interesting areas in a graph that might be good candidates for new hypotheses. It instead computes the importance of a node within the full graph. While an importance factor surely is a very interesting measure within graphs consisting of documents linked through references, for example the Web or a co-author citation network (Ding et al., 2009), it adds little to no value to directed networks consisting of biomedical entities and their interactions with each other in the original form. The point here is that we are usually not looking for the nodes that have the most impact or importance in the network, i.e. the nodes that are the most popular (because research at those nodes is already high) but instead the nodes that are different. PageRank has, however, its place in biomedicine. An adaptation of the algorithm was performed by Weston et al. (2006), called RankProp, and used to enhance upon protein ranking algorithms, such as PSI-BLAST (Altschul et al., 1997). It cannot, however, be used for the intended use case of this work as we are not looking for important nodes ranked by some schema but instead looking for areas that seem under- or overrepresented and therefore appear interesting to a scientist. Important nodes, such as those detected by PageRank, while probably also being interesting are probably already under heavy research and therefore seem inadequate for our purposes.

6.3 The FriendSuggest and WrongBob algorithms

In the study "Suggesting (More) Friends Using the Implicit Social Graph" by Roth et al. (2011), the authors analyzed the mechanisms, how people communicate with each other through email and how this knowledge can be used to support a user in finding correct or wrong recipients of an email draft. The core idea is that despite of users loosely grouping their contacts into (overlapping) groups, such as *family, co-workers, friends, ...*, those groups can also be implicitly detected from the underlying communication graph based on emails with multiple recipients. Subsequently, those graphs can be used to detect whether emails go consistently to members of the same group or not, making it possible to identify unintentionally wrong or left-out recipients. In their study, they present a new algorithm that is able to suggest potential contacts (nodes) from the communication patterns between them (links or relations).

The core of their idea is the so-called edge-weight interactions rank that calculates weights to edges according to three criteria: *frequency* of interaction over time, *recency* of an interaction, and *direction* of an interaction (passive less important than active). It is defined as

$$IR = w_{\text{out}} \sum_{i \in I_{\text{out}}} \left(\frac{1}{2}\right)^{\frac{1}{\lambda}(t_{\text{now}} - t(i))} + \sum_{i \in I_{\text{in}}} \left(\frac{1}{2}\right)^{\frac{1}{\lambda}(t_{\text{now}} - t(i))} \tag{6.2}$$

where w_{out} is the relative importance of outgoing vs incoming evidences, I_{out} and I_{in} are the sets of outgoing and incoming interactions, t_{now} is the current time and $t(i)$ the timestamp of interaction i to account for the recency. λ describes the half-life of the importance of an interaction, meaning an interaction from now contributes 1, interaction from (now $-\lambda$) contributes ½ and so on, conforming with an exponential decay rate.

This function now basically represents a score that is assigned to each of the neighbors of a particular node, in their case the email contacts of a particular user. For each user, the contacts can be ordered with this function according to the criteria. Contacts of a user can now be compared to each other. A comparison of the contacts of different users is, however, not possible with that method.

Roth et al. made use of those observations and introduced two features to Google's GMail service[80]: FriendSuggest and WrongBob. The *FriendSuggest* algorithm uses the interactions rank knowledge in order to suggests additional recipients for an email that could have been left out. More mathematically, the algorithm tries to find nodes in a network of recipients that makes the group of recipients more coherent (see Alg. 6.1). *WrongBob* on the other hand performs the opposite of that. It tries to find nodes whose removal would make the recipient group more coherent (see Alg. 6.2). Colloquially, this means the identification of wrong recipients within a list. In GMail it is mainly used to discriminate between people in an address book with the same name in order to not accidentally sending the email to the wrong recipient.

Now, in order to transfer this solution to the biomedical domain, we only need to abstract these algorithms away from email communication networks and address books to the biomedical interaction

[80] http://mail.google.com

function EXPANDSEED($u, \mathcal{S}$):

1: $\mathcal{G} \leftarrow$GETGROUPS(u)
2: $\mathcal{F} \leftarrow \emptyset$
3: **for each** group $g \in \mathcal{G}$ **do**
4: **for each** contact $c \in g, c \notin \mathcal{S}$ **do**
5: **if** $c \notin \mathcal{F}$ **then**
6: $\mathcal{F}[c] \leftarrow 0$
7: **end if**
8: $\mathcal{F}[c] \overset{+}{\leftarrow}$UPDATESCORE($c, \mathcal{S}, g$)
9: **end for**
10: **end for**
11: **return** F

Input: u the user, $\mathcal{S}$ the seed
Returns: $\mathcal{F}$ the friend suggestion

Algorithm 6.1: The ExpandSeed algorithm. The core routine of the FriendSuggest algorithm suggests contacts that expand a particular seed, given a user's contact groups. Basically, for each contact in a user's address book, the score is calculated for the inclusion in a user's groups. Adopted from Roth et al. (2011).

function WRONGBOB(u, L):

1: scoreMax $\leftarrow 0$
2: wrongRecipient $\leftarrow$ *null*
3: suggestedContact $\leftarrow$ *null*
4: **for each** contact $c_i \in L$ **do**
5: seed $\leftarrow L \backslash c_i$
6: results $\leftarrow$ EXPANDSEED(u, seed)
7: **if** $c_i \in$ results **then**
8: **continue**
9: **end if**
10: **for each** contact $c_j \in$ results **do**
11: **if** ISSIMILAR(c_i, c_j) **and** score(c_j) $>$ scoreMax **then**
12: scoreMax $\leftarrow$ score(c_j)
13: wrongRecipient $\leftarrow c_i$
14: suggestedContact $\leftarrow c_j$
15: **end if**
16: **end for**
17: **end for**
18: **return** {wrongRecipient, suggestedContact}

Input: u the user, L a list of recipients of an email
Returns: a pair $\{c, s\}$ where c is a contact $\in L$, s is a suggested contact to replace c

Algorithm 6.2: The WrongBob algorithm. The algorithm basically retrieves the group members for each recipients in the email draft. These group members are then compared to the other recipients with a similarity score. Adopted from Roth et al. (2011).

network from Excerbt. For an automated hypothesis generation this would then enable two different scenarios:

1. Suggest new nodes that could be included in a model (which is usually a subgraph of the complete literature graph) with an adaptation of FriendSuggest.
2. Suggest nodes that seem to be unlikely interaction partners according to the given context with an adaptation of the WrongBob algorithm. Those unlikely nodes could be, according to the definition of interestingness (see Section 6.1), very useful hints for next research targets.

Taken together, these tools could be a very useful help for any scientist working with models in order to find additional research targets However, in order to be able to use those systems for the biomedical domain, a few adaptations to the algorithms involved have to be discussed.

6.4 A Clustering Algorithm to Find Implicit Neighbor Groups

The main point of adapting FriendSuggest and WrongBob to the biomedical domain is the problem of acquiring the initial groups in the GetGroups method used in Alg. 6.1. While obtaining implicit groups off a node's neighbors is a straight-forward approach in social networks, such as email, Twitter, or Facebook, it is difficult for biomedical networks because those networks usually do not have *this* kind of implicit underlying network structure that could be used.

One idea for an underlying, implicit network within biomedical networks might be the co-mentioning of the same relation in different articles. Consider a network as the one generated by Excerbt, where nodes are biomedical entities and edges are their relations found by semantic analysis within text. Each edge has the evidences of their citations attached. An implicit network could be constructed by only following links that are from the same publication, i.e. for any source node N, we group all neighbors that have evidences based on the same publication together. With this approach, a direction and a time dimension would both be available. However, while being an implicit underlying network structure, it cannot be used to predict groups in the desired schema. A publication could be used as the equivalent of a message. There are, however, several problems associated with this. For one, a single publication defines multiple edges across the whole literature network and is not necessarily limited to only one source node that is currently reviewed. This means that any document would have to contribute multiple relations with at least one identical interaction partner instead of relations between arbitrary interaction partners. A second point is that the InteractionsRank's frequency attribute cannot be applied because as far as the Excerbt network suggests, literature is (usually) not written through simple copying statements (i.e. describing the same relation over and over) and writing them time and again in new publications. Instead, most interactions usually have only one or two literature evidences attached as evidenced in Figure 5.6 (page 111). This basically gives the main reason, why we cannot use the implicit network structure as suggested by Roth et al. (2011) for biomedical networks.

Figure 5.7 (page 112) shows the influence of publications to the distribution of the number of evidences within the size-1 neighborhood network of any particular node. As can be seen, the peak with nearly 65.000 nodes is at 0.2. Colloquially, this means that most publications contribute more than one

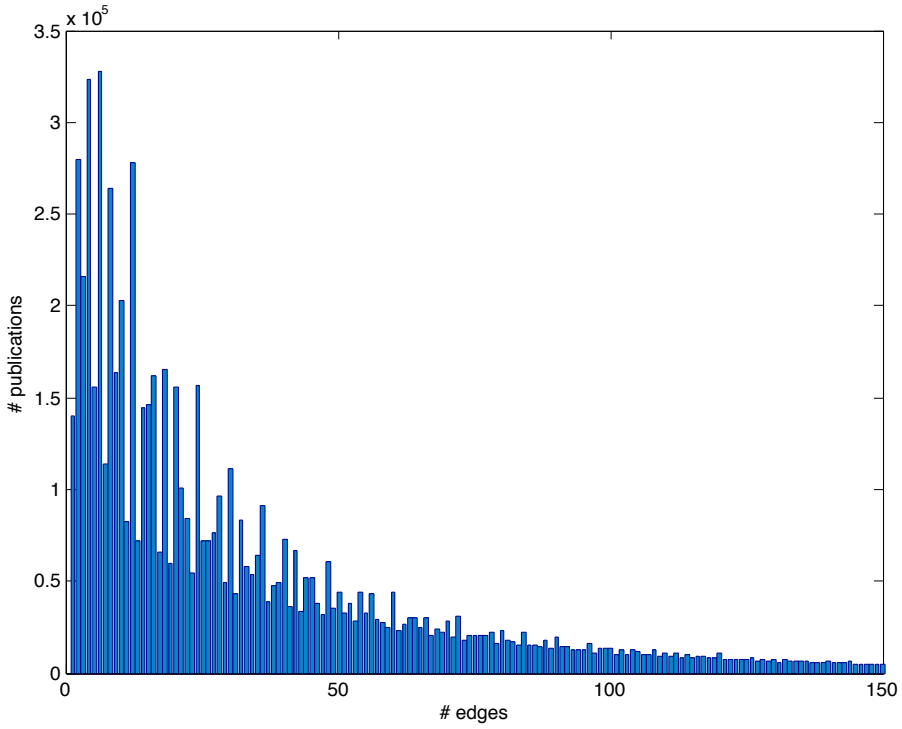


Figure 6.3: Histogram of the number of publications and their relations. The histogram shows that the number of relations compared against the number of publications with that number of relations decreases exponentially with more relations. The different peaks suggest that the distribution is made up of two different distributions: one for Pubmed MEDLINE and one for Pubmed Central (PMC). PMC articles are full-text articles and therefore longer. Pubmed citations are only the abstract and consist of approximately 10 sentences each. The high peaks at the lower end of the number of edges therefore seem to represent mainly Pubmed abstracts. Data as at 2011/12/31.

evidence to the same node. It does, however, not state whether these contributions are to the same target node or to different nodes.

Having argued why the message- or publication-based implicit network is not of particular applicability within biomedical networks, the fact that the main point for using this particular implicit network in the original publication (Roth et al., 2011) was that from any user's perspective *only* the egocentric neighborhood network of size one was available. In biomedical networks, however, we have the full network structure at our disposal and can use it accordingly.

The main point now is to find a method to compute those neighborhood groups (GetGroups) in order to use them as input for the FriendSuggest and WrongBob methods.

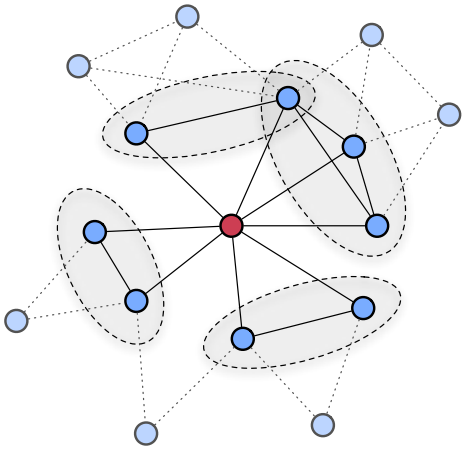


Figure 6.4: Graphical overview of the GetGroups method. The algorithm tries to identify neighbor groups (gray, dashed encircled) of a central node (red). For that it includes two things: direct links between the neighbors (black lines) and indirect links between the neighbors (grayed, dotted lines). For each pair of neighbors, a connectivity score S is computed and groups are identified by thresholds.

I devised an algorithm for the GetGroups method that is (principally) based on the entire network structure, not just the first neighborhood of the starting node (outlined in Algorithm 6.3). The algorithm takes into account the connectivity of the neighbors among each other in order to determine their group association. A simplification of the algorithm takes the direct neighborhood of the start node and the neighbor's neighbors into account (size two egocentric neighborhood network). With this network, the neighborhood is fuzzy-clustered according to the connectivity of the neighbors themselves with inclusion of indirect connections between them. This is illustrated in Figure 6.4. Those neighbors are then ranked and a score is computed for each pair of neighbors. If this score is higher than a specified threshold t, both nodes are put into the same group. To compute this score, I devised the following equation:

$$S_{i,j} = \sum_{p \in \mathcal{P}(n_i, n_j)} e^{-\lambda(L(p)-1)} w(p) \tag{6.3}$$

The scoring function contains an exponential decay function. Basically, for each path between two arbitrary nodes within the neighborhood network (without the original node!) a score is computed that decays with the length of the path. All scores for all paths p between two nodes n_i and n_j are then

added and written as the overall score for that node pair. $L(p)$ gives the length of the path p, i.e. the number of intermediary edges. If two nodes are directly linked to each other in a specific path, they get a score of 1, if $\lambda = 1$ and $w(p) = 1$. The weighting function $w(p)$ can be used to account for example for the number of evidences for each edge in the paths and/or the directionality of the links[81].

The output of the algorithm is a list of fuzzy (overlapping) neighbor groups that can be used directly for the ExpandSeed method.

6.4.1 Evaluation with the CFinder Community Detection Method

In order to prove whether the algorithm produces significantly correct results, we implemented the FriendSuggest with an adapted scoring function and evaluated it with the CFinder method by Palla et al. (2005) (see Section 2.1.2). We used the standard method as outlined in Algorithm 6.3 and compared results to the CFinder method with standard settings. Correct, in this case, means that the detected neighbor groups have a high overlap with the communities found by CFinder. Test data was the co-authorship network as introduced by Palla et al. (2008). We decided that co-authorship networks are sufficiently able to prove the correctness of the method as their composition and global properties do not significantly differ from other biological networks, such as protein-protein interactions.

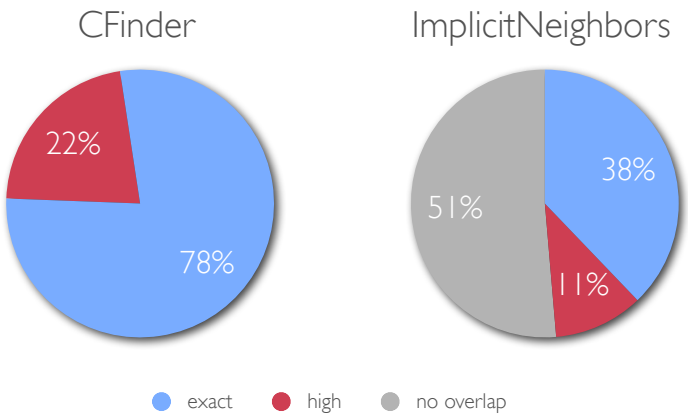


Figure 6.5: CFinder vs. implicit neighbor clustering.

Figure 6.5 shows the results of sample egocentric subnetworks of size two for five arbitrarily chosen nodes from the test network. After performing both methods, it turned out that 78% of the (local) clusters that were detected by CFinder were also detected by the Implicit Neighbor Group Finder method. 22% of these clusters had a high overlap in the participating nodes of a group. Turned the other way around, 51% of the groups output by the Implicit Neighbor method have not been found by CFinder. This investigation, however, does not take into account thresholds, therefore in order to reproduce the results of the CFinder package, probably only the thresholds would have to be adjusted. This was, however, not performed, since the main focus of the new method is to detect edges that are

[81] The weighting function $w(p)$ could be defined to weight paths differently whether the links in the path are orthodromic ($\rightarrow\rightarrow$, $\leftarrow\leftarrow$) or antidromic ($\leftarrow\rightarrow$, $\rightarrow\leftarrow$)

function GETGROUPS(u):

1: $N \leftarrow$GETNEIGHBORHOOD($u, 2$)
2: $S \leftarrow null$
3: **for each** direct neighbors $n_i, n_j \in N_{direct}$, $n_i \neq n_j$ **do**
4: $\mathcal{P} \leftarrow$GETPATHS(n_i, n_j, N)
5: **if** $(n_i, n_j) \notin \mathcal{P}$ **then**
6: **continue**
7: **end if**
8: $S_{i,j} \leftarrow \sum_{p \in \mathcal{P}(n_i,n_j)} e^{-\lambda(\text{LENGTH}(p)-1)} w(p)$
9: **end for**
10: $S' \leftarrow$NORMALIZE(S)
11: $\mathcal{G} \leftarrow \emptyset$
12: **for each** line $i \in S'$ **do**
13: **for each** column $j \in S'_i$ **do**
14: **if** $S'_{i,j} >$ threshold t **then**
15: $\mathcal{G} \xleftarrow{+} \{n_i, n_j\}$
16: **end if**
17: **end for**
18: **end for**
19: $\mathcal{G}' \leftarrow$COMBINENEIGHBORPAIRSTOGROUPS($\mathcal{G}$)
20: **return** neighbor groups $\mathcal{G}'$

Input: The size-2 neighborhood network of a particular node n
λ decay constant, currently set to 1
$w(p)$ path weighting function, currently defined as: $w(p) = 1$
t the threshold when a link is significant enough
Returns: A set of overlapping groups of neighbors of node u

Algorithm 6.3: The implicit neighbor clustering method. First step is to calculate the scoring matrix S by the given equation in line 8 (Equation 6.3). The Normalize method normalizes the whole matrix by the highest value in order to make individual entries comparable to each other. Then, for each line in the scoring matrix S', node pairs with a score higher than a threshold t are identified and combined to neighbor pairs. The CombineNeighborPairsToGroups method combines the list of node-pairs $\mathcal{G}$ to a list of groups.

"interesting" according to the criteria from the beginning of this chapter and not to perform perfectly correct clustering of nodes.

6.4.2 A MapReduce Implementation of the Implicit Neighbor Clustering Method

The biomedical literature network extracted with Excerbt is very large in terms of nodes and edges as we have seen in Chapter 5. The GroupFinder method as outlined above is not a feasible approach to detect groups on a large scale for all nodes in a network. As the method basically aims to assign a score to hypothetical edges that are not (yet) manifested in the network, the number of edges to check can increase dramatically with an increased network size. A complete network with n nodes has $n(n-1)$ edges at maximum for the directed case or $n(n-1)/2$ in case of an undirected network.

In collaboration with a bachelor's student of mine, Tanzeem Haque (Haque, 2012), I devised an algorithm to compute scores for all not-manifested edges in a network with an adapted GroupFinder method that works with egocentric subnetworks of size two (see Algorithm 6.4). The basic principle of the algorithm is to generate all possible edges between any two nodes and look for a scoring depending on the neighbors of the two participating nodes of the posed edge. It works similar to the clustering coefficient MapReduce algorithm and represents an iterative approach for the score-computation for the edges. The algorithm is currently limited to the egocentric neighborhood of depth two, but a generalization to egocentric networks of arbitrary depth s would be possible by adding more iterations to the MapReduce stack as in the case with a parallel breadth-first search.

The algorithm basically consists of three reducer classes, since for all involved steps a list of values is needed. The MapReduce framework for Hadoop does unfortunately not support the calling of multiple reducers directly after the other and always needs a new mapper in between. Since these mappers don't perform any computation, they have been omitted in the outline of the algorithm. The first reducer has a node and its neighborhood of size one and emits an edge for every combination of tuples of nodes within the neighborhood and the original nodes together with the original central node. The second reducer then gets these edges as the key and a list of all central nodes that this edge is a neighbor of as the values. Then a score for this edge is calculated according to the number of elements in the central nodes list. The interesting part is that if the nodes of the edge also occur in the list of the central nodes, the edge actually exists in the network and can then be handled accordingly. In this case it gets a higher score as to discriminate those later with the two thresholds introduced in the previous section. As a last step in the third reducer, we have scores for all possible edges in the network, we need to normalize the scores within a specific egocentric subnetwork of a single node as to make the scores of the overall network comparable to each other.

The main difference of this MapReduce implementation to the GroupFinder method is that we cannot feasibly determine whether a neighboring edge of a central node actually exists in the network or not. Therefore, a global view of all possible interactions has to be created and with it we then can determine the existence according to some criteria (reducers 1 and 2). The overall result of the MapReduce application is a new network that has the same nodes as the original network but is augmented with scores for every possible edge. This network can optionally be filtered according to the same threshold criteria as the traditional variant of the algorithm. Since we now have scores for every possible edge, also a new weighting according to the number of nodes that suggested this edge can be created.

function REDUCE1(node n, list of neighbors N):

1: **for each** nodes $s, t \in N$ **do**
2: emit($\{n, s\}$, n) emit actual edges
3: emit($\{n, t\}$, n)
4: emit($\{s, t\}$, n) emit hypothetical edges
5: **end for**

function REDUCE2(edge $\{s, t\}$, list of neighbors N):

6: edge? $\leftarrow s \overset{?}{\in} N \wedge t \overset{?}{\in} N$ test whether this edge actually exists
7: **for each** nodes $n \in N \setminus \{s, t\}$ **do**
8: score $\leftarrow 0.25 \times$ sizeof(N) $- 3$
9: **if** edge? **then**
10: emit(n, 1+score)
11: **else**
12: emit(n, score)
13: **end if**
14: **end for**

function REDUCE3(node n, list of edge-score tuples L):

15: L' = NORMALIZESCORESBYHIGHEST(L)
16: **for each** edge-scores $e, s \in L$ **do**
17: emit(e, $\{n, s\}$)
18: **end for**
19: score $\leftarrow$ GETSCOREOFWEIGHTEDEDGE(E)
20: emit(e, score)

Input: A network represented as nodes and their adjacency lists
Returns: A network where interesting additional edges are augmented with an hypotheticity group affiliation score

Algorithm 6.4: GroupFinder for neighborhood size two in MapReduce. In the first step, all actual and hypothetical edges that can be reached from the input node are computed and emitted into the next phase. The second step (Reduce2) has all these edges as input, determines which ones exist and computes the scores. For all nodes, the surrounding edge scores are emitted. In the third step all the scores are combined into a single score for a specific edge.

In order to implement a FriendSuggest feature, where based on models and given input groups new members (nodes) are predicted, one would only have to apply the threshold function and use the initial groups to extract nodes that have a hypothetical edge with a score above the threshold to any of the nodes in the group and predict this as a new member. However, the implementation as presented here also allows another application that takes the large-scale network analysis approaches more into consideration, as can be seen in the next section.

6.5 HypoGen – A Method to Generate Suggestions from Implicit Neighbors

The score-augmented network from the previous section can now be used in order to generate suggestions according to the interestingness criteria as defined in the beginning of this chapter (Section 6.1). The basic idea is to search for edges that do not exist in the original network but have a hypothetical score near a threshold to nodes that belong to local groups but because of the score lower than the thresholdwere not considered part of the group. Figure 6.6 illustrates this approach with the hypothetical edge highlighted in red.

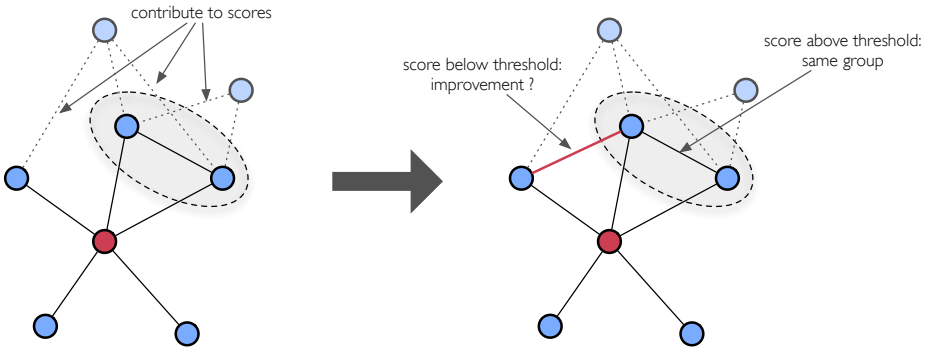


Figure 6.6: The HypoGen method. Coming from a network and a clustering of the neighbors for each node, edges are predicted that could improve the overall network structure by including additional nodes into existing groups.

In order to perform this on a large scale, that is for all nodes in Excerbt's literature network, we simply need to compute all possible local groups and determine for each scored edge, how many neighbors (relative) have predicted this edge based on their neighborhood, i.e. the number of red nodes as in Figure 6.6. Put differently, we would run the group finding algorithm once for every node in the network. For each non-existing edge with a score above a certain threshold we would then count the number of neighboring nodes that predicted that edge above the score threshold.

The first interestingness definition, "the absence of something that would be expected" can now be solved by looking for particular edges that do not exist in the original network and have a high score (near one). According to our scoring schema, these edges were predicted by neighboring nodes with a high probability. At this step, the number of individual predictions for that edge can be considered or not, depending on the number of high-scoring predicted edges.

The second problem, "the presence of something that does not fit right" can also be tackled by the approach. An edge that exists in the original network but has a low score near zero can be classified as an edge that is unlikely because of the structure of the network. A low score basically states that the edge is of limited probability because of the inherent structure of the neighboring nodes. If, however, the edge existed in the original network, this must have some meaning leading to an area in the network that can be considered interesting.

When performing the HypoGen method on a large scale with the MapReduce algorithm (6.4) from Section 6.4.2, the definition of the correct thresholds is left until the last step. This enables us to define the thresholds in dependence of the actual nodes of a particular subnetwork. For the standard version of the algorithm, two thresholds were defined: 0.2 and 0.6. These worked very well for the co-authorship network used for the evaluation with the CFinder method (Section 6.4.1). In this case, the global histogram for the full network and the local histogram for a particular section of the network for the distribution of scores is a useful measure.

In the scope of Tanzeem Charu's bachelor's thesis (Haque, 2012), we evaluated the MapReduce approach on two networks: the yeast protein-protein interactome (PPI) (Yu et al., 2008) and a subnetwork of Excerbt's literature graph, constructed from Parkinson's disease. The subnetwork was created by retrieving all gene neighbors of Parkinson's disease and all gene neighbors of those genes, i.e. the egocentric gene neighborhood of depth two. Scores were computed for edges between the direct neighbors of Parkinson's disease only and in the case of the PPI network for all edges.

Figure 6.7 shows a section of the yeast PPI network with edges highlighted according to score. A red edge has a score over 0.6, a green edge between 0.2 and 0.6 and a grey edge below 0.2. Dashed edges are predicted by the method. An example predicted edge is in the top middle of the figure. Between the nodes YOL005C and YOR341W the HypoGen algorithm put an edge with a score above 0.6. A closer examination of the two proteins shows that their surrounding neighbors are tightly interconnected. Based on that observation a hypothetical edge is predicted for the two proteins. Whether this makes actually sense or not in a biological way is now up to a biologist to determine. The method gives, however, a good insight into areas of the network that might be worth exploring.

As a more concrete example, Figure 6.8 shows the output of the HypoGen method on the egocentric gene network of Parkinson's disease. Noticeable is a clear distribution of scores on the lower (red) end of the spectrum, near 0.0. The hypothetical edge with the highest score was posed between *actin* and *COX2*. The term actin represents any actin-binding proteins that crosslinks actin filaments. A quick search with the indirect query function on the web interface resulted in a list of 299 intermediate genes that could be responsible for an interaction between the two proteins. The amount of intermediate genes supports the hypothesis that there might be an interaction of any kind between the two proteins. At this point, I want to make clear again that the method does not state a regulating or otherwise directly influencing interaction between two nodes in the network. Rather the statement is that these might somehow be involved in similar processes. The large amount of intermediate genes/proteins seems to suggest that there might indeed be cause for further investigations. Determining the real extent of the interaction between any such two entities is up to an experimental biologist who is actively working in the field of Parkinson's disease and might accept or refute the stated hypothesis.

The PPI network contains 1092 nodes and 1318 edges, the Parkinson's disease network contains 14901 nodes and 45269 edges. Both networks have a scale-free attribute, despite the fact that the latter has

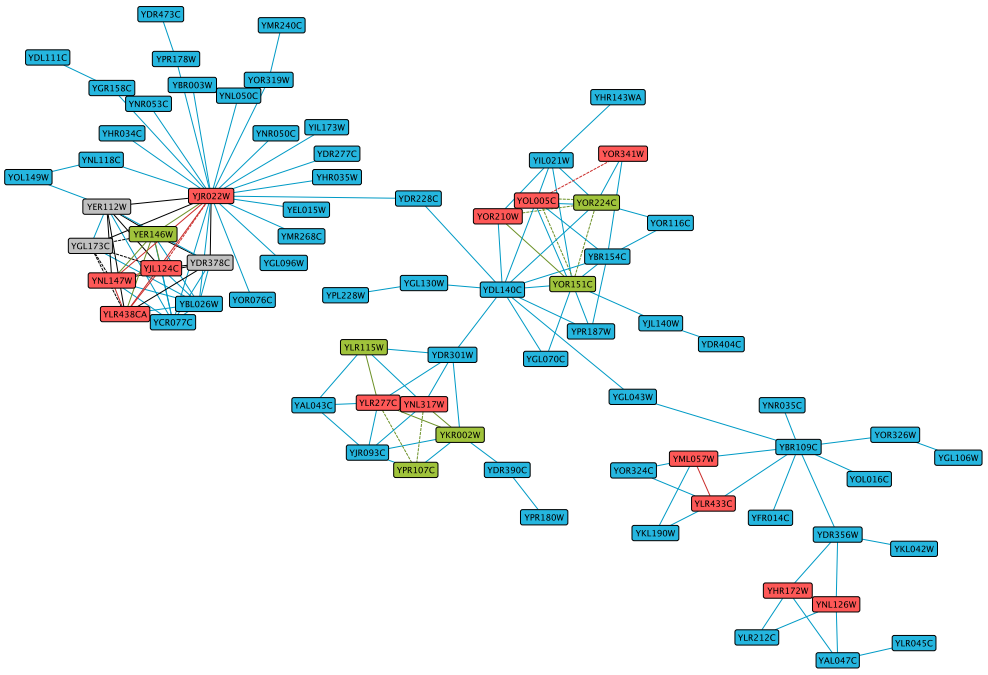
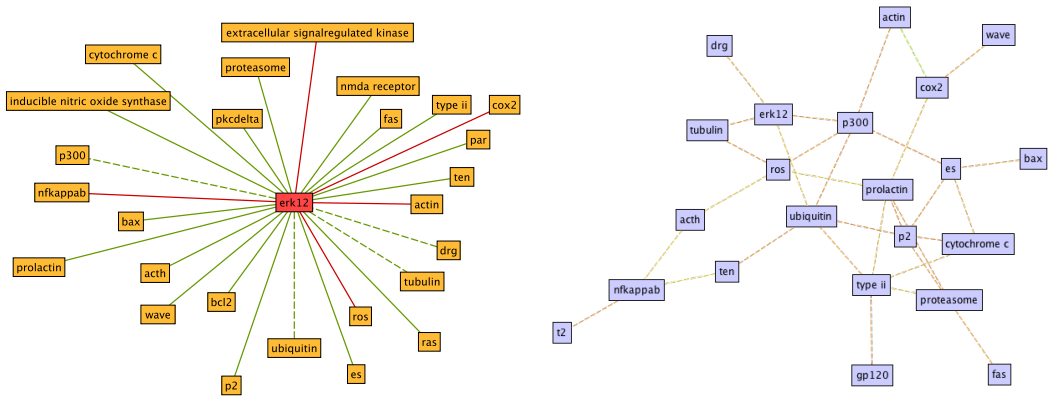


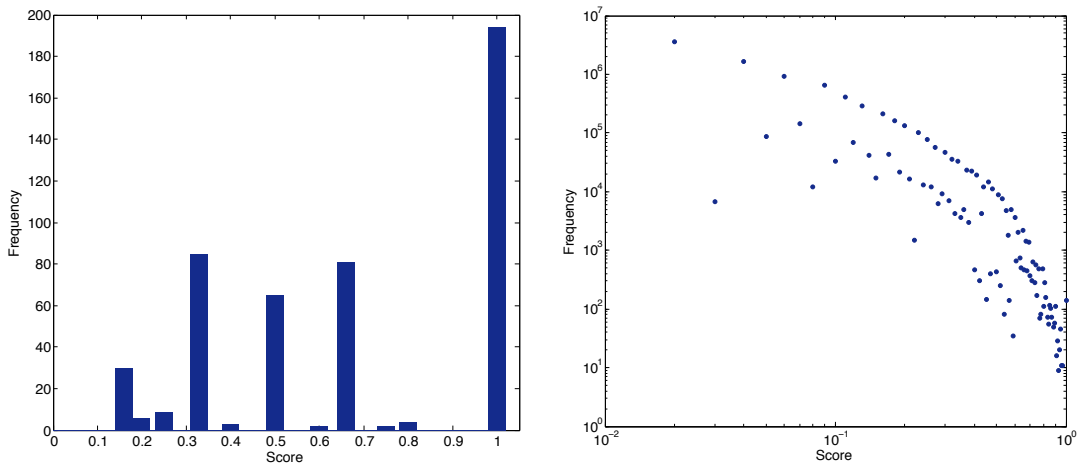
Figure 6.7: Section of the yeast PPI network from Yu et al. (2008) with annotated edges. Red edges: score greater than 0.6, green edges: score greater 0.2, grey edges: score below 0.2. Solid edges are present in the original network, dashed edges are predicted edges.



(a) (b)

Figure 6.8: Hypothetical edges in Parkinson's disease's egocentric gene network. (a) Predicted and existing edges for ERK1/2. Green edges have a low score, red edges have a high score. Dashed edges do not exist in the original network. (b) Edges are colored from red to green, with fully green edges having a score of 1.0 and fully red edges a score of 0.0. The edges are all dashed because they all do not exist in the original PD network. Most edges are colored in red tone since the scores all were at the lower end.

far more edges than nodes. For a better comparison we have plotted the histograms of the scores for the two networks in Figure 6.9. Interestingly, while the PPI network's score distribution seems to be random with the maximum number of edges for score 1.0, the PD network's score distribution seems to exhibit exponential decrease in frequency towards bigger scores. One reason might be that the yeast PPI network clearly is a very well understood network that has not many areas left to explore, while the PD network still is in the beginnings of a deeper understanding. Another reason might be that the PPI network is much smaller and therefore contains less predicted interactions and the plot therefore has less significance.



(a) Yeast PPI network

(b) Parkinson's disease network

Figure 6.9: Score frequency histograms for the HypoGen method. (a) shows the scores for the yeast protein-protein interaction network. (b) shows the scores for the egocentric Parkinson's disease network. Interesting to note is that while the latter exposes an exponentially decreasing probability distribution, the former PPI network seems to distribute scores randomly. Data as at 2012/07/31.

The HypoGen approach does, however, have one drawback. If the algorithm would be executed repeatedly on a specific network and predicted edges with a score above a certain threshold would be considered as fact, where the output of the algorithm is the input of the algorithm of the next phase, the network would eventually become a complete network. The nodes would stay the same, but at some point every node would have an edge to every other node in the network. While this would normally not be performed it outlines the problem: Biological, organically evolved networks, most of the times have a hierarchical scale-free property. With this approach, the prediction of new edges ignores the fundamental principle of scale-free networks: preferential attachment. The hub-and-spoke model of the original network is ignored and new edges are predicted regardless of an influence to the degree and clustering coefficient distributions. However, as the method does not state "this is a missing edge!" for a predicted edge but rather "this area in the network looks promising, you might want to have a look at it", the method can still be successfully used in order to gain a better understanding of particular processes within network-like structures. Basically, the HypoGen method finds over- and underspecified areas in the network that might be worth to examine more precisely.

For the future, a version of the algorithm that factors in preferential attachment could be developed. The scoring method would have to be adapted to include the degree and clustering coefficient of the

participating nodes in order to lower the score for nodes with a small degree and increase the score for nodes with a higher degree, proportional to a degree distribution consistent with preferential attachment, such as the power law distribution.

6.6 Summary

This chapter introduced a definition for interesting areas in network structures. Modified from an algorithm developed originally for the Google Mail platform, I have created a method to identify local groups of neighbors within the scope of a particular node in a network, the implicit neighbor group finder method. The method was then adapted to work with data stored in graph form in Hadoop MapReduce in order to predict group membership scores for all edges in the complete Excerbt graph. With this data, I have then developed the HypoGen method that uses the implicit neighbors in order to predict potential edges in a network that hint at potential future research areas because of an underspecification of a particular region in the network.

Obviously, this can only be a first step. As an outlook for the automated knowledge inference from network data, there already exist some clustering methods implemented in MapReduce. For the future, such methods could be used to pre-cluster the full literature network into distinct scientific subfields, e.g. research particular to specific diseases, such as cancer research or diabetes research. With that, the runtime of the HypoGen method could be drastically decreased for individual queries from individual scientists which would then enable an integration into the web interface. The algorithm to predict the group affiliation scores, specifically the scoring function could furthermore be enhanced with an adjustment for the directionality of edges in the network. It could involve the relation type in some way as this would certainly improve the results, not to also predict the function between the two but to enable a more differentiated scores for the different types. This would also include the addition of negated and hypothetical interactions between entities, which would further enhance the quality of a score.

The implications and advantages of the method are manifold for biomedical research. An insight, based on the structure of the network, which areas of interaction are significantly under- or overrepresented seems to be a great resource for anyone performing research in that particular area.

Some of the methods and results of this section, particularly the HypoGen method, have been presented at a talk at the International Conference on Systems Biology 2012 in Toronto, Canada (Wachinger et al., 2012).

7 Conclusion and Outlook

Within the scope of this thesis I have introduced a new text mining system that is capable of handling massive and exponentially growing amounts of biomedical publications, so called big data literature. For this, I have rewritten the text mining system Excerbt, a knowledge extraction framework based on semantic role labeling and relation extraction and proposed a range of new methods to perform the relevant tasks under the viewpoint of scalability to such exponentially increasing data sets. From a technological point of view, I have created a big data architecture with drastically changed components of the system. I have particularly put my focus on scalable approaches to dictionary-based named entity recognition with millions of lexical entries in millions of documents. The newly created system is now capable of efficiently querying more than 70 million unique entity names with the possibility to incrementally query either new entities against a corpus of text data or new text against an existing corpus of entity names.

On a smaller scale, I have extended the application regarding relation extraction with respect to more sensitivity and greater details in the type of extractable knowledge. In the first instance, I have investigated how scientists communicate within publications based on previous research and derived rules in order to improve the extraction of relations. The Excerbt text mining system has been altered to allow more rules for relation extraction and I included several other improvements such as a limitation of the parts in a sentence that can be part of a relation to reduce the amount of false positives.

The architecture of the new system is based on the Hadoop MapReduce framework, a solution to efficiently batch-process large amounts of data by a form of a divide-and-conquer approach. For data storage I have implemented access to the big data database HBase that can interoperate with MapReduce on a low level. Additionally, I have created a sophisticated web interface and query language to retrieve the extracted and contained knowledge from the Excerbt database and to intuitively display it in a web browser. The query language allows to specify the different parts of a relation as a query and additionally enables the support of indirect queries which can return intermediate entities between two given entities that interact with both of them in any specifyable type.

I have also demonstrated the capabilities of the new system by performing a number of analyses within the scope of the scalability of the system to big data. I have introduced methods to work with n-gram data of all texts in the Pubmed MEDLINE and Pubmed Central corpora in order to demonstrate a new awareness of how biomedical language is structured and to infer novel understanding at a massive scale.

Excerbt's knowledge data is represented as a huge graph, specifically a topic map, in terms of nodes and edges. I have developed corresponding big data algorithms in order to work with that graph. I have devised adapted algorithms for computing degree and clustering coefficient distributions of huge graphs and argued that these findings now support network analysis on a large scale as we are used to

from a Systems Biology perspective due to the network being hierarchical and having the scale-free property.

With these changes, Excerbt is now a text mining system that supports three of the four text mining approaches. It supports co-occurrence, relation-based, and statistical text mining, but not yet sentiment analysis. While the co-occurrence part has not been described extensively in this thesis as it was not an integral constituent of the new system, the other approaches enabled new insights into the composition of biomedical literature. At this time, a first outlook into a system of the future would be the inclusion of the fourth type, sentiment analysis. Albeit the fact that the positivity or negativity of statements adds little value to a relation, such as a protein-protein interaction or similar, the derivable assessment of impartiality, whether or not a certain statement is described objectively or subjectively, can be a great indicator for various weighting criteria and enable a subsequent better modeling of biomedical complex systems.

Finally, I have also dedicated a chapter to the analysis of big graph data in order to find suggestive edges within the network for further research. The presented method allows scientists to assess certain regions within the knowledge graph of Excerbt according to various criteria that define interest, such as the presence or absence of particular edges that should differ from the current situation because of the neighboring graph structure. The method can be easily generalized to work with other networks. Results have shown that these methods can clear up the cluttered view of a huge graph associated to particular areas of research and provide scientists with reasonable suggestions upon which they could base further work.

A promising area of future research is definitely the adaptation of the named entity recognition module to include heuristic approaches. The next generation Excerbt system is built in a way that single modules can easily be extended for other methods but not affecting the overall results negatively. A reason to include heuristic approaches is certainly not the fact that found hits are wrong, but more owed to the fact that ontologies usually cannot keep up with the increasing number of newly invented terms. A combination of ontology-based and heuristic methods surely would be best for the system.

Excerbt is also currently solely based on relations extracted from within a single sentence. An adaptation of the relation extraction process beyond the sentence boundaries is therefore desirable and will open up even more opportunities for knowledge extraction and scientific research. The integration of anaphora resolution, connecting sentences working with identifiers such as “it”, “they”, or “the disease” to the previous actual declaration of the subject, comes to mind as well as the resolution of abbreviations within introductory passages.

Despite this work being in large parts technology-driven, the opportunities with an existing such system for research are manifold. The basic task of any text mining system and Excerbt in particular is to extract knowledge from textual data, which means to structure the unstructured. A resource that is on the one hand capable of processing big data volumes efficiently and has access to the structured knowledge from text resources on the other hand has nearly unlimited potential with respect to novel methods of knowledge management. Besides the obvious integration of multiple information resources, such as experimentally validated protein-protein interaction data, micro-array experiments, or statistical data, other resources could be overlayed as well. Citation networks from publications, who cites whom, could be overlapped with the actual text passages and their extracted knowledge. Combined with a time-related view on the data, this would provide an extremely helpful way to

track, how research is done and how research progresses in particular fields. To expand on this idea, this could in theory enable the appropriate authorities to better track how research is going and subsequently better allot personal and financial resources.

Such a feature is also highly related to the identification of research paths in a network. Not only for authorities, but also for scientist, the identification of significant developments over time could open up new insights into the research process. A collection of methods and materials, for example, that work or do not work would represent an invaluable resource. Tracking which decisions in what type of research work best could be collected and would subsequently enable a completely new approach to research.

Regarding some prospects of the here presented approches that are nearer in the future lie specifically in the Systems Biology area. The possibilities to automatically infer models of biomedical complex systems, such as diseases, can be of great help in understanding the etiology of the respective process. Paired with powerful frontends to the system that enable researchers an interactive annotation of deduced networks and models offer formidable tools in objectifying research, enabling new insights and a better tracking of progress.

Another, more concrete area of future work is to analyze a single publication with resprect to the contained knowledge and compare its statements against the background network of previous publications or any other text related data set. Such a method could identify, whether the contents of the new publication are in unison or in contradiction to existing literature, whether it provides new insights or simply repeats already known information. This could lead the way to an automated system to peer review new publications and pave the way for publications with better quality in the long-term. With the Excerbt system presented in this thesis, building such a system would be straight-forward. The ad-hoc analysis module for a single document would have to be extended to allow an assessment of found relations, not only a display. For this to work, research would have to be put into the development of such assessment functionality.

On the front of large-scale knowledge graph analysis methods, a next step could be to cluster the Excerbt graph according to network structure. This would on the one hand provide an even more efficient technological solution to the distribution of the network across a cluster of machines and on the other hand then enable an in-depth analysis of the intricacies of interrelated subfields of research. That is to say, different research area, for example between two different disease types, could be investigated combined to infer novel insights into the way scientists perform research or how these field are interacting with each other despite having completely different scientific communities.

To summarize this thesis again, I have basically introduced two things within its scope: First of all, I devised a way to create a version of a biomedical text mining system that is exponentially scalable. I have identified common problems, especially when taking dictionary-based named entity recognition and an efficient graph-based representation of knowledge into account and provided solutions in the respective sections. This showed that it is possible with sophisticated tools to tackle the exponential data increase in a time- and hardware-efficient manner.

Second of all, I have introduced a novel big-data enabled methods that use graph-based knowledge in order to predict areas of interest within a network that hint at situations that are worth to be further examined by scientists working in that field. This shows that again with adequate methods it is possible

to use large-scale knowledge effectively in order to gain fine-grained insights by analyzing all available data.

If one thing should have been clarified throughout this thesis, it is that only now — that big data enabled methods on biomedical literature have been developed — research can be performed that takes all of the available data into account. This enables an abundance of new ideas and techniques to perform on data sets, completely in accordance with the notion behind Systems Biology, which will lead to an even better extraction and usage of knowledge from unstructured data in the future.

A List of POS Tags and Semantic Roles

Table A.1 gives an overview of the Penn Treebank part-of-speech tags as used in various publications and throughout this thesis. The descriptions were taken from Jurafsky and Martin (2009)

Table A.2 gives an overview of the different semantic roles as used in Palmer et al. (2005) and Collobert and Weston (2007). Here, only the modifier arguments (`ARGM-`) are given since the descriptions of the regular arguments (`ARG0` – `ARG6`) are highly dependent on the verb.

Tag	Description	Example
CC	coordinating conjunction	and, but, or
CD	cardinal number	one, two, three
DT	determiner	a, the
EX	existential “there”	there
FW	foreign word	mea culpa
IN	preposition or subordinating conjunction	of, in, by
JJ	adjective	yellow
JJR	adjective, comparative	bigger
JJS	adjective, superlative	wildest
LS	list item marker	1, 2, One
MD	modal	can, should
NN	noun, singular or mass	llama, snow
NNS	noun, plural	llamas
NNP	proper noun, singular	IBM
NNPS	proper noun, plural	Carolinas
PDT	predeterminer	all, both
POS	possessive ending	’s
PRP	personal pronoun	I, you, he
PRP$	possessive pronoun	your, one’s
RB	adverb	quickly, never
RBR	adverb, comparative	faster
RBS	adverb, superlative	fastest
RP	participle	up, off
SYM	symbol	+, %, &
TO	“to”	to
UH	interjection	ah, oops
VB	verb, base form	eat
VBD	verb, preterite (past tense)	ate
VBG	verb, gerund	eating
VBN	verb, past participle	eaten
VBP	verb, non-third-person singular present tense	eat
VBZ	verb, third-person singular present tense	eats
WDT	wh-determiner	which, that
WP	wh-pronoun	what, who
WP$	possessive wh-	whose
WRB	wh-adverb	how, where

Table A.1: List of Penn Treebank part-of-speech tags. Punctuation marks have been omitted. Adapted from Jurafsky and Martin (2009)

Argument	Name	Description
ARGM-CAU	cause	syntactic elements starting with phrases like *because* or as result of indicating the reason for an action, questions starting with *why*
ARGM-DIR	direction	elements containing source and goal of a movement following a clear path. If no clear path, ARGM-LOC is used
ARGM-DIS	discourse	constituents like *also*, *however* or *as well* connecting a sentence to a preceding one
ARGM-EXT	extent	numerical adjuncts and comparatives stating the amount of change occurring from an action
ARGM-LOC	location	physical and abstract locations indicating *where* some action takes place
ARGM-MNR	manner	answers to *how* an action is performed questions
ARGM-ADV	general purpose	modifiers of the event structure of a verb; ARGM-ADV modifies the entire sentence, ARGM-MNR only modifies the
ARGM-MOD	modal verb	Modals: **will**, **may**, **can**, **must**, **shall**, **might**, **should**, **could**, **would**. Phrasal modals **going (to)**, **have (to)** and **used (to)**
ARGM-NEG	negation marker	negation elements such as *not*, *-n't*, *never*, *no longer* and others
ARGM-PNC	purpose	motivation for the action, but not the cause; typical example: clauses beginning with **in order to**
ARGM-TMP	time	*when* an action took place, also includes adverbs of frequency, adverbs of duration, order and repetition
ARGM-PRD		an adjunct of a predicate can in itself be capable of carrying some predicate structure and thus is labeled with ARGM-PRD.
ARGM-REC		Reflexives and reciprocals such as **himself**, **itself**, **themselves** are comprised in the reciprocal tag.

Table A.2: List of modifier arguments. Numbered arguments ARGx have been omitted since their meaning highly depends on the actual verb. Adapted from Hummel (2011), Palmer et al. (2005), and Collobert and Weston (2007).

B List of Relation Extraction Rules

In the following, a list of rules is given that can be used in the relation extraction steps of the Excerbt text mining system. Comparison and correlation rules have been analyzed in the context of Hummel (2011), causal/temporal rules in Strache (2012). Rules have been converted into computer-readable format.

Comparison Rules

```
:in .. :than := in .. than, in .. compared to, for .. in comparison with
<basis> := has_POS_tag <noun>
<comparative> := more, less, higher, lower, ...

1. ARG0 matches <entity>E1 &&
   ARG1 matches (<comparative>C * <basis>B * <entity>E2)
2. ARG1 matches <basis>B &&
   ARG3 matches (<comparative>C :in <entity>E1 :than <entity>E2)
3. ARG1 matches (<comparative>C <basis>B :in <entity>E1 :than <entity>E2)
4. ARG1 matches <basis>B &&
   ARGM_MNR matches <comparative>C &&
   ARG3 matches (:in <entity>E1 than <entity>E2)
5. ARG1 matches <entity>E1 &&
   ARGM_MNR matches <comparative>C &&
   ARG3 matches (<basis>B :than <entity>E2)
6. ARG0 matches <entity>E1 &&
   ((ARG1 matches <basis>B || ARG2 matches <basis>B)) &&
   ((ARGM_ADV matches <entity>E2 || ARGM_MNR matches <entity>E2 ||
     ARGM_PNC matches <entity>E2))
7. ARG1 matches <basis> &&
   ARG2 matches <entity>E1 &&
   ((ARGM_ADV matches <entity>E2 || ARGM_MNR matches <entity>E2 ||
     ARGM_PNC matches <entity>E2))

OUTPUT: E1 --[compared to]--> E2; context={basis:B, direction:C}
```

Correlation Rules

```
:correlate := correlate, relate, associate, connect
:correlation := correlation, relationship, connection, association
:between .. :and := between .. and, among .. and, of .. with
:with := with
:for := for
:to := to

1. ((ARG0 matches <entity>E1 || ARG1 matches <entity>E2)) <entity>E1 &&
    VERB is_type :correlate && ARG2 matches <entity>E2
2. ARG0 matches <entity>E1 &&
   VERB is_type :correlate && ARG2 matches <entity>E2
3. ARG1 matches <entity>E2 E1 &&
   VERB is_type <correlate> && ARG0 matches <entity>E2
4. ((ARG1 matches (:correlation :between <entity>E1 :and <entity>E2) ||
   ARGM_MNR matches (:correlation :between <entity>E1 :and <entity>E2) ))
5. ARG1 matches (<entity>E1 :correlation :with <entity>E2)
6. ARG0 matches <entity>E1 &&
   ARG1 matches (:correlation :with <entity>E2)
7. ARG1 matches (:correlation :with <entity>E1) &&
   ARG3 matches (:for <entity>E1)
8. ARG1 matches (:correlation :with <entity>E1) &&
   ARGM_ADV matches (:for <entity>E1)
9. ARG1 matches <entity>E1 &&
   ((ARG0 matches (:correlation :with <entity>E2) ||
     ARG2 matches (:correlation :with <entity>E2)))
0. ARG1 matches (<entity>E1 :correlation :to <entity>E2) ||
   ARGM_MNR matches (<entity>E1 :correlation :to <entity>E2)

OUTPUT: E1 [correlates with] E2
```

Causal/Temporal Rules

```
1. ARG1 not_exists &&
   ((ARGM_CAU matches <entity>E1 || ARGM_TMP matches <entity>E1)) &&
   ARG0 matches <entity>E2
2. ARG0 not_exists &&
   ((ARGM_CAU matches <entity>E1 || ARGM_TMP matches <entity>E1)) &&
   ARG1 matches <entity>E2

OUTPUT: E1 [verb-relationtype] E2
```

List of Abbreviations

In the following, various abbreviations and acronyms used throughout this thesis will be written out in full. Order is alphabetical.

Acronym	Full form
ACID	Atomicity, Consistency, Isolation, Durability
AWS	Amazon Web Services
BASE	basically available, soft state, eventually consistent
CAP	Consistency, Availability, Partition-Tolerance
CC	Clustering Coefficient
CPU	Central Processing Unit
CSV	Comma-Separated Values
DBMS	Database Management System
DNA	Deoxyribonucleic acid
DSL	Domain-Specific Language
EC2	Elastic Compute Cloud
EQL	Excerbt Query Language
ES	ElasticSearch
FDA	Food and Drug Administration
HDFS	Hadoop Distributed File System
HSC	Hematopoietic Stem Cell
I/O	Input/Output
IaaS	Infrastructure as a Service
IR	Information Retrieval
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LAN	Local Area Network
NER	Named Entity Recognition
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
NoSQL	not only SQL
OC	Ovarian Cancer
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PaaS	Platform as a Service
PAS	Predicate-Argument Structure
PCR	Polymerase Chain Reaction
PD	Parkinson's Disease

Acronym	Full form
POS	Part-of-Speech
PPI	Protein-Protein Interaction
RAM	Random-Access Memory
RDBMS	Relational Database Management System
REST	Representational State Transfer
RNA	Ribonucleic acid
SaaS	Software as a Service
SENNA	Semantic Extraction via a Neural Network Architecture
SIDER	Side Effect Resources
SNP	Single Nucleotide Polymorphism
SQL	Structured Query Language
SRL	Semantic Role Labeling
STI	Sentence Token Index
TM	Text Mining
TSV	Tab-Separated Values
VLAN	Virtual Local Area Network
XML	Extensible Markup Language

List of Figures

List of Tables

List of Algorithms

Bibliography

D. J. Abadi. Hadoop's tremendous inefficiency on graph data management (and how to avoid it), 2011. URL http://dbmsmusings.blogspot.com/2011/07/hadoops-tremendous-inefficiency-on.html#.

S. Agarwal and H. Yu. Automatically Classifying Sentences in Full-Text Biomedical Arti-cles into Introduction, Methods, Results and Discussion. *Bioinformatics*, Sept. 2009. doi: 10.1093/bioinformatics/btp548. URL http://dx.doi.org/10.1093/bioinformatics/btp548.

K. Ahmed and G. Moore. An Introduction to Topic Maps. *The Architecture Journal, Microsoft*, (5), July 2005.

R. Albert, H. Jeong, and A. Barabasi. Error and attack tolerance of complex networks. *Nature*, 406(6794): 378–382, July 2000. doi: 10.1038/35019019. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=10935628&retmode=ref&cmd=prlinks.

S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–3402, Sept. 1997. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=9254694&retmode=ref&cmd=prlinks.

D. Alur, J. Crupi, and D. Malks. *Core J2EE Patterns: Best Practices and Design Strategies*. Sun Microsystems Press. Prentice Hall / Sun Microsystems Press, 1st edition edition, 2001. ISBN 0130648841. URL http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html.

S. Ananiadou, S. Pyysalo, J. Tsujii, and D. B. Kell. Event extraction for systems biology by text mining the literature. *Trends Biotechnol*, 28(7):381–390, July 2010. doi: 10.1016/j.tibtech.2010.04.005. URL http://dx.doi.org/10.1016/j.tibtech.2010.04.005.

C. Anderson. Is Zipf's Law just a statistical glitch?, Oct. 2006. URL http://www.longtail.com/the_long_tail/2006/09/is_zipfs_law_ju.html.

N. O. Andrews and E. A. Fox. Recent Developments in Document Clustering, Oct. 2007. URL http://eprints.cs.vt.edu/archive/00001000/01/docclust.pdf.

M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, and G. Lee. Above the Clouds: A Berkeley View of Cloud Computing. Department of Electrical Engineering and Computer Sciences. *University of California at Berkeley, Berkeley, CA*, 2009. URL http://scholar.google.com/scholar?q=related:BatbPFwPFqQJ:scholar.google.com/&hl=en&num=30&as_sdt=0,5.

M. Arnold, M. L. Hartsperger, H. Baurecht, E. Rodríguez, B. Wachinger, A. Franke, M. Kabesch, J. Winkelmann, A. Pfeufer, M. Romanos, T. Illig, H.-W. Mewes, V. Stümpflen, and S. Weidinger. Network-based SNP meta-analysis identifies joint and disjoint genetic features across common human diseases. *BMC Genomics*, 13 (1):490, Sept. 2012. doi: 10.1186/1471-2164-13-490. URL http://www.biomedcentral.com/1471-2164/13/

490/abstract.

K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and E. Paulson. Efficient processing of data warehousing queries in a split execution environment. *Proceedings of the 2011 international conference on Management of data*, pages 1165–1176, 2011. URL http://dl.acm.org/citation.cfm?id=1989447.

B. Balasundaram and S. Butenko. Network Clustering. In *Analysis of Biological Networks*. Wiley, 2008.

S. Banon. ElasticSearch - Road to a Distributed Search Engine. In *Berlin Buzzwords*, Aug. 2011. URL http://www.elasticsearch.org/videos/2011/08/09/road-to-a-distributed-searchengine-berlinbuzzwords.html.

A.-L. Barabási and Z. N. Oltvai. Network biology: understanding the cell's functional organization. *Nat Rev Genet*, 5(2):101–113, Feb. 2004. doi: 10.1038/nrg1272. URL http://dx.doi.org/10.1038/nrg1272.

E. Barnett. Facebook cuts six degrees of separation to four. *The Telegraph*, Nov. 2011.

T. Barnickel. *Large Scale Knowledge Extraction From Biomedical Literature based on Semantic Role Labeling*. PhD thesis, July 2009.

T. Barnickel, J. Weston, R. Collobert, H.-W. Mewes, and V. Stümpflen. Large Scale Application of Neural Network Based Semantic Role Labeling for Automated Relation Extraction from Biomedical Texts. *PLoS One*, 4(7): e6393, July 2009. doi: 10.1371/journal.pone.0006393. URL http://dx.doi.org/10.1371%2Fjournal.pone.0006393.

S. Bethard, Z. Lu, J. H. Martin, and L. Hunter. Semantic role labeling for protein transport predicates. *BMC Bioinformatics*, 9:277, 2008. doi: 10.1186/1471-2105-9-277. URL http://dx.doi.org/10.1186/1471-2105-9-277.

P. E. Black. Dictionary of Algorithms and Data Structures, 2004. URL http://xlinux.nist.gov/dads/.

C. Blake. Beyond genes, proteins, and abstracts: Identifying scientific claims from full-text biomedical articles. *J Biomed Inform*, 43:173—-189, Nov. 2009. doi: 10.1016/j.jbi.2009.11.001. URL http://dx.doi.org/10.1016/j.jbi.2009.11.001.

P. Blohm. *Supersemantics for Knowledge Extractions*. PhD thesis, TU München, Jan. 2014.

J. Bollen, H. Mao, and X.-J. Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, abs/1010.3003, 2010. URL http://arxiv.org/abs/1010.3003.

D. Borthakur. HDFS Architecture Guide, 2010. URL http://hadoop.apache.org/hdfs/docs/current/hdfs_design.html.

R. S. Boyer and J. S. Moore. A fast string searching algorithm. *COMMUNICATIONS OF THE ACM*, 20(10):762–772, Oct. 1977. doi: 10.1145/359842.359859. URL http://portal.acm.org/citation.cfm?doid=359842.359859.

M. M. Bradley and P. J. Lang. Affective norms for English words (ANEW): Instruction manual and affective ratings. Technical Report C-1, 1999. URL http://a.parsons.edu/~spani621/thesis/context/ANEW.pdf.

E. A. Brewer. Towards robust distributed systems (abstract). In *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*. ACM Request Permissions, July 2000. doi: 10.1145/343477.343502. URL http://portal.acm.org/citation.cfm?id=343477.343502&coll=DL&dl=

GUIDE&CFID=82476000&CFTOKEN=23366389.

C. S. Brinegar. Mark Twain and the Quintus Curtius Snodgrass letters: A statistical test of authorship. *Journal of the American Statistical Association*, 58(301):85–96, 1963. doi: 10.2307/2282956.

A. Broder. From Query Based Information Retrieval to Context Driven Information Supply. In *grupowebupfes*. Yahoo, June 2006. URL http://grupoweb.upf.es/workshop/slides/fws_broder.pdf.

V. Bush. As We May Think. *Atlantic Monthly*, 176(1):641–649, Mar. 1945. doi: 10.1145/227181.227186. URL http://www.theatlantic.com/doc/194507/bush.

F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. *OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006.*, 2006. URL http://labs.google.com/papers/bigtable.html.

H. Chen and B. M. Sharp. Content-rich biological network constructed by mining PubMed abstracts. *BMC Bioinformatics*, 5:147, Oct. 2004. doi: 10.1186/1471-2105-5-147. URL http://dx.doi.org/10.1186/1471-2105-5-147.

C. Chew and G. Eysenbach. Pandemics in the age of Twitter: content analysis of Tweets during the 2009 H1N1 outbreak. *PLoS One*, 5(11):e14118, 2010. doi: 10.1371/journal.pone.0014118. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=21124761&retmode=ref&cmd=prlinks.

E. H. Chi. Who Knows? Searching for Expertise on the Social Web. *COMMUNICATIONS OF THE ACM*, 55(4):110, Apr. 2012. doi: 10.1145/2133806.2133829. URL http://portal.acm.org/citation.cfm?doid=290159.290160.

E. F. Codd. A relational model of data for large shared data banks. *COMMUNICATIONS OF THE ACM*, 13(6):377–387, June 1970. doi: 10.1145/362384.362685. URL http://portal.acm.org/citation.cfm?doid=362384.362685.

A. M. Cohen, W. R. Hersh, C. Dubay, and K. Spackman. Using co-occurrence network structure to extract synonymous gene and protein names from MEDLINE abstracts. *BMC Bioinformatics*, 6:103, 2005. doi: 10.1186/1471-2105-6-103. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=15847682&retmode=ref&cmd=prlinks.

J. Cohen. Graph Twiddling in a MapReduce World. *Computing in Science and Engineering*, 11:29–41, 2009. doi: http://doi.ieeecomputersociety.org/10.1109/MCSE.2009.120.

K. B. Cohen and L. Hunter. Getting started in text mining. *PLoS Comput Biol*, 4(1):e20, Jan. 2008. doi: 10.1371/journal.pcbi.0040020. URL http://dx.doi.org/10.1371/journal.pcbi.0040020.

R. Collobert and J. Weston. Fast Semantic Extraction Using A Novel Neural Network Architecture. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 560–567. Association for Computational Linguistics, June 2007.

R. Collobert and J. Weston. Fast Semantic Extraction Using a Neural Network Architecture. pages 1–11, Aug. 2008.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa. Natural Language Processing (almost) from Scratch. *CoRR*, abs/1103.0398, 2011. URL http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/de//pubs/archive/35671.pdf.

J. Constine. How Big Is Facebook's Data? 2.5 Billion Pieces Of Content And 500+ Terabytes Ingested Every Day, Aug. 2012. URL http://techcrunch.com/2012/08/22/how-big-is-facebooks-data-2-5-billion-pieces-of-content-and-500-terabytes-ingested-every-day/.

T. Converse, R. M. Kaplan, B. Pell, S. Prevost, L. Thione, and C. Walters. Powerset's Natural Language Wikipedia Search Engine. In *Wikipedia and Artificial Intelligence: An Evolving Synergy, Papers from the 2008 AAAI Workshop*, 2008. URL http://www.aaai.org/Library/Workshops/2008/ws08-15-013.php.

D. Dahlmeier and H. T. Ng. Domain Adaptation for Semantic Role Labeling in the Biomedical Domain. *Bioinformatics*, Feb. 2010. doi: 10.1093/bioinformatics/btq075. URL http://dx.doi.org/10.1093/bioinformatics/btq075.

D. Das and S. Petrov. Unsupervised part-of-speech tagging with bilingual graph-based projections. In *HLT '11: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, June 2011. URL http://portal.acm.org/citation.cfm?id=2002472.2002549&coll=DL&dl=GUIDE&CFID=79764158&CFTOKEN=14524558.

J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *OSDI'04*, 2004.

M. deHaaff. Sentiment Analysis, Hard But Worth It! — CustomerThink, Mar. 2010. URL http://www.customerthink.com/blog/sentiment_analysis_hard_but_worth_it.

I. Derényi, G. Palla, and T. Vicsek. Clique percolation in random networks. *Phys Rev Lett*, 94(16):160202, Apr. 2005. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=15904198&retmode=ref&cmd=prlinks.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec. 1959. doi: 10.1007/BF01386390. URL http://www.springerlink.com/index/10.1007/BF01386390.

L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659, New York, NY, USA, 2004. ACM. ISBN 1-58113-874-1. doi: http://doi.acm.org/10.1145/1031171.1031289. URL http://doi.acm.org/10.1145/1031171.1031289.

Y. Ding, E. Yan, A. Frazho, and J. Caverlee. PageRank for ranking authors in co-citation networks. *Journal of the American Society for Information Science and Technology*, 60(11), Nov. 2009. URL http://portal.acm.org/citation.cfm?id=1656289.1656306&coll=DL&dl=GUIDE&CFID=138408822&CFTOKEN=52827053.

S. Even. *Graph Algorithms*. Cambridge University Press, 2nd edition edition, Sept. 2011. ISBN 978-0521736534.

Eventbrite. NOSQL meetup, June 2009. URL http://nosql.eventbrite.com/.

Hive, 2008. Facebook Data Team.

D. Farber. On-demand computing: What are the odds?, Nov. 2002. URL http://www.zdnet.com/news/on-demand-computing-what-are-the-odds/296135.

R. T. Fielding. Architectural styles and the design of network-based software architectures, 2000. URL http://ps-2.kev009.com/rs6000/redbook-cd/SG246158.PDF.

M. Fontoura, M. Gurevich, V. Josifovski, and S. Vassilvitskii. Efficiently Encoding Term Co-Occurrences in Inverted Indexes. In *20th ACM Conference on Information and Knowledge Management*, 2011.

M. S. Forman, J. Q. Trojanowski, and V. M.-Y. Lee. Neurodegenerative diseases: a decade of discoveries paves the way for therapeutic breakthroughs. *Nature medicine*, 10(10):1055–1063, Oct. 2004. doi: 10.1038/nm1113. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=15459709&retmode=ref&cmd=prlinks.

S. Fortunato. Community detection in graphs. *Physics Reports*, 486,:75–174, 2010.

I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10, 2008. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4738445.

J. Gantz and D. Reinsel. Extracting Value from Chaos, June 2011. URL http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf.

L. M. Garshol and G. Moore. Topic Maps — Data Model, June 2008. URL http://www.isotopicmaps.org/sam/sam-model/2008-06-03/.

J. Geelan. Twenty-One Experts Define Cloud Computing, Jan. 2009. URL http://cloudcomputing.sys-con.com/node/612375.

L. George. *HBase: The Definitive Guide*. O'Reilly, 2011a. URL http://www.hbasebook.com.

L. George. Apache HBase – Eine quelloffene BigTable Implementierung. *Java Magazin*, 11:39–45, 2011b.

S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. *19th Symposium on Operating Systems Principles, Lake George, NY*, 19:29–43, 2003.

S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2), June 2002. doi: 10.1145/564585.564601. URL http://portal.acm.org/citation.cfm?id=564585.564601&coll=DL&dl=GUIDE&CFID=82476000&CFTOKEN=23366389.

D. Gildea and D. Jurafsky. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3), Sept. 2002. URL http://www.mitpressjournals.org/doi/abs/10.1162/089120102760275983.

K.-I. Goh, M. E. Cusick, D. Valle, B. Childs, M. Vidal, and A.-L. Barabási. The human disease network. *Proceedings of the National Academy of Sciences of the United States of America*, 104(21):8685–8690, May 2007. doi: 10.1073/pnas.0701361104. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=17502601&retmode=ref&cmd=prlinks.

Google. Google history. URL http://www.google.com/intl/en/about/company/history.html.

W. R. Gowers. *A manual of diseases of the nervous system*. Blakiston, Philadelphia, 2 edition, 1892. URL http://openlibrary.org/books/OL24406899M/A_manual_of_diseases_of_the_nervous_system.

A. Granelli-Piperno. SRC-related proto-oncogenes and transcription factors in primary human T cells: modulation by cyclosporin A and FK506. *Journal of autoimmunity*, 5 Suppl A:145–158, Apr. 1992. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=1380242&retmode=ref&cmd=prlinks.

S. F. Greenbury, I. G. Johnston, M. A. Smith, J. P. K. Doye, and A. A. Louis. The effect of scale-free topology on the robustness and evolvability of genetic regulatory networks. *Journal of theoretical biology*, 267(1): 48–61, Nov. 2010. doi: 10.1016/j.jtbi.2010.08.006. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=20696172&retmode=ref&cmd=prlinks.

N. Gross. Literature-based Inference and Analysis of Connections between Genes and Metabolism-related Side Effects. Master's thesis, Dec. 2011.

F. E. Grubbs. Procedures for Detecting Outlying Observations in Samples. *Technometrics*, 11(1):1–21, Feb. 1969.

T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5 (2), June 1993. URL http://portal.acm.org/citation.cfm?id=173743.173747&coll=DL&dl=GUIDE&CFID=136443565&CFTOKEN=38314240.

P. Gupta. Twitter Engineering: Cassovary: A Big Graph-Processing Library, Mar. 2012. URL http://engineering.twitter.com/2012/03/cassovary-big-graph-processing-library.html.

S. T. Haque. *Massive-scale network analysis for automated biomedical synonym resolution*. PhD thesis, TU München, 2012.

N. Harmston, W. Filsell, and M. Stumpf. Which species is it? Species-driven gene name disambiguation using random walks over a mixture of adjacency matrices. *Bioinformatics*, Nov. 2011. doi: 10.1093/bioinformatics/btr640. URL http://dx.doi.org/10.1093/bioinformatics/btr640.

W. Hersh. Evaluation of biomedical text-mining systems: lessons learned from information retrieval. *Briefings in Bioinformatics*, 6(4):344–356, Dec. 2005. URL http://bib.oxfordjournals.org/cgi/content/abstract/6/4/344.

A. Hitchcock. Google's BigTable, Oct. 2005. URL http://andrewhitchcock.org/?post=214.

T. Hoff. Drop ACID And Think About Data. May 2009a. URL http://highscalability.com/drop-acid-and-think-about-data.

T. Hoff. Neo4j - A Graph Database that Kicks Buttox, June 2009b. URL http://highscalability.com/neo4j-graph-database-kicks-buttox.

R. Hoffmann and A. Valencia. A gene network for navigating the literature. *Nat Genet*, 36(7):664, July 2004. doi: 10.1038/ng0704-664. URL http://dx.doi.org/10.1038/ng0704-664.

C. Holst. A/B Testing: Begin with a Problem and Hypothesis, Feb. 2011. URL http://baymard.com/blog/ab-testing-problem-and-hypothesis.

J. Huang, D. J. Abadi, and K. Ren. Scalable SPARQL Querying of Large RDF Graphs. In *Proceedings of the VLDB Endowment, Vol. 4, No. 11*, 2011. URL http://cs-www.cs.yale.edu/homes/dna/papers/sw-graph-scale.pdf.

B. Hummel. Analysis of Semantic Role Annotation of Biomedical Literature for Sensitivity Improvement of Automatically Inferred Systems Biological Models. Master's thesis, Ludwig Maximilians University and Technical University Munich, May 2011.

L. Hunter and K. B. Cohen. Biomedical language processing: what's beyond PubMed? *Mol Cell*, 21(5):589–594, Mar. 2006. doi: 10.1016/j.molcel.2006.02.012. URL http://dx.doi.org/10.1016/j.molcel.2006.02.012.

M. Indelicato. Scalability Strategies Primer: Database Sharding, Dec. 2008. URL http://blog.maxindelicato.com/2008/12/scalability-strategies-primer-database-sharding.html.

T. Ivarsson. NoSQL for Dummies. In *Miracle Open World 2010*, pages 1–53, Denmark, 2010. URL http://www.slideshare.net/thobe/nosql-for-dummies/download.

G. Jaganadh. Opinion Mining and Sentiment Analysis, May 2012. URL http://www.csi-india.org/c/document_library/get_file?uuid=a02abd49-2c26-4af9-91f0-ce16fe21f2f7&groupId=10616.

J. Jarman. Combining Natural Language Processing and Statistical Text Mining: A Study of Specialized Versus Common Languages, 2011. URL http://scholarcommons.usf.edu/cgi/viewcontent.cgi?article=4361&context=etd.

D. Joyner, M. Van Nguyen, and N. Cohen. *Algorithmic Graph Theory*. Jan. 2012. URL http://code.google.com/p/graphbook/.

D. Jurafsky and J. H. Martin. *Speech and Language Processing*. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Pearson Education Inc., 2009. ISBN 0135041961.

N. Klarlund and M. Riley. Word n-grams for cluster keyboards. In *TextEntry '03: Proceedings of the 2003 EACL Workshop on Language Modeling for Text Entry Methods*. Association for Computational Linguistics, Apr. 2003. URL http://portal.acm.org/citation.cfm?id=1628195.1628202&coll=DL&dl=GUIDE&CFID=147238308&CFTOKEN=42836571.

Y. Kogan, N. Collier, S. Pakhomov, and M. Krauthammer. Towards semantic role labeling & IE in the medical literature. *AMIA Annu Symp Proc*, pages 410–414, 2005.

A. Kollegger. Neo4j 1.3 "Abisko Lampa" M04 - Size really does matter, Mar. 2011. URL http://blog.neo4j.org/2011/03/neo4j-13-abisko-lampa-m04-size-really.html.

M. Krallinger, A. Valencia, and L. Hirschman. Linking genes to literature: text mining, information extraction, and retrieval applications for biology. *Genome Biol*, 9 Suppl 2:S8, 2008. doi: 10.1186/gb-2008-9-s2-s8. URL http://dx.doi.org/10.1186/gb-2008-9-s2-s8.

M. Kuhn, M. Campillos, I. Letunic, L. J. Jensen, and P. Bork. A side effect resource to capture phenotypic effects of drugs. *Mol Syst Biol*, 6:343, 2010. doi: 10.1038/msb.2009.98. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=20087340&retmode=ref&cmd=prlinks.

B. Kuo, T. Hentrich, B. Good, and M. Wilkinson. Tag clouds for summarizing web search results. *Proceedings of the 16th international conference on World Wide Web*, pages 1203–1204, 2007. URL http://dl.acm.org/citation.cfm?id=1242766.

E. S. Lander, L. M. Linton, B. Birren, C. Nusbaum, M. C. Zody, J. Baldwin, K. Devon, K. Dewar, M. Doyle, W. FitzHugh, R. Funke, D. Gage, K. Harris, A. Heaford, J. Howland, L. Kann, J. Lehoczky, R. LeVine, P. McEwan, K. McKernan, J. Meldrim, J. P. Mesirov, C. Miranda, W. Morris, J. Naylor, C. Raymond, M. Rosetti, R. Santos, A. Sheridan, C. Sougnez, N. Stange-Thomann, N. Stojanovic, A. Subramanian, D. Wyman, J. Rogers, J. Sulston, R. Ainscough, S. Beck, D. Bentley, J. Burton, C. Clee, N. Carter, A. Coulson, R. Deadman, P. Deloukas, A. Dunham, I. Dunham, R. Durbin, L. French, D. Grafham, S. Gregory, T. Hubbard, S. Humphray, A. Hunt, M. Jones, C. Lloyd, A. McMurray, L. Matthews, S. Mercer, S. Milne, J. C. Mullikin, A. Mungall, R. Plumb, M. Ross, R. Shownkeen, S. Sims, R. H. Waterston, R. K. Wilson, L. W. Hillier, J. D. McPherson, M. A. Marra, E. R.

Mardis, L. A. Fulton, A. T. Chinwalla, K. H. Pepin, W. R. Gish, S. L. Chissoe, M. C. Wendl, K. D. Delehaunty, T. L. Miner, A. Delehaunty, J. B. Kramer, L. L. Cook, R. S. Fulton, D. L. Johnson, P. J. Minx, S. W. Clifton, T. Hawkins, E. Branscomb, P. Predki, P. Richardson, S. Wenning, T. Slezak, N. Doggett, J. F. Cheng, A. Olsen, S. Lucas, C. Elkin, E. Uberbacher, M. Frazier, R. A. Gibbs, D. M. Muzny, S. E. Scherer, J. B. Bouck, E. J. Sodergren, K. C. Worley, C. M. Rives, J. H. Gorrell, M. L. Metzker, S. L. Naylor, R. S. Kucherlapati, D. L. Nelson, G. M. Weinstock, Y. Sakaki, A. Fujiyama, M. Hattori, T. Yada, A. Toyoda, T. Itoh, C. Kawagoe, H. Watanabe, Y. Totoki, T. Taylor, J. Weissenbach, R. Heilig, W. Saurin, F. Artiguenave, P. Brottier, T. Bruls, E. Pelletier, C. Robert, P. Wincker, D. R. Smith, L. Doucette-Stamm, M. Rubenfield, K. Weinstock, H. M. Lee, J. Dubois, A. Rosenthal, M. Platzer, G. Nyakatura, S. Taudien, A. Rump, H. Yang, J. Yu, J. Wang, G. Huang, J. Gu, L. Hood, L. Rowen, A. Madan, S. Qin, R. W. Davis, N. A. Federspiel, A. P. Abola, M. J. Proctor, R. M. Myers, J. Schmutz, M. Dickson, J. Grimwood, D. R. Cox, M. V. Olson, R. Kaul, C. Raymond, N. Shimizu, K. Kawasaki, S. Minoshima, G. A. Evans, M. Athanasiou, R. Schultz, B. A. Roe, F. Chen, H. Pan, J. Ramser, H. Lehrach, R. Reinhardt, W. R. McCombie, M. de la Bastide, N. Dedhia, H. Blöcker, K. Hornischer, G. Nordsiek, R. Agarwala, L. Aravind, J. A. Bailey, A. Bateman, S. Batzoglou, E. Birney, P. Bork, D. G. Brown, C. B. Burge, L. Cerutti, H. C. Chen, D. Church, M. Clamp, R. R. Copley, T. Doerks, S. R. Eddy, E. E. Eichler, T. S. Furey, J. Galagan, J. G. Gilbert, C. Harmon, Y. Hayashizaki, D. Haussler, H. Hermjakob, K. Hokamp, W. Jang, L. S. Johnson, T. A. Jones, S. Kasif, A. Kaspryzk, S. Kennedy, W. J. Kent, P. Kitts, E. V. Koonin, I. Korf, D. Kulp, D. Lancet, T. M. Lowe, A. McLysaght, T. Mikkelsen, J. V. Moran, N. Mulder, V. J. Pollara, C. P. Ponting, G. Schuler, J. Schultz, G. Slater, A. F. Smit, E. Stupka, J. Szustakowski, D. Thierry-Mieg, J. Thierry-Mieg, L. Wagner, J. Wallis, R. Wheeler, A. Williams, Y. I. Wolf, K. H. Wolfe, S. P. Yang, R. F. Yeh, F. Collins, M. S. Guyer, J. Peterson, A. Felsenfeld, K. A. Wetterstrand, A. Patrinos, M. J. Morgan, P. de Jong, J. J. Catanese, K. Osoegawa, H. Shizuya, S. Choi, Y. J. Chen, J. Szustakowki, and International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, Feb. 2001. doi: 10.1038/35057062. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=11237011&retmode=ref&cmd=prlinks.

S. Leberknight. Polyglot Persistence - Blogs at Near Infinity, Aug. 2008. URL http://www.nearinfinity.com/blogs/scott_leberknight/polyglot_persistence.html.

K. Leetaru. Culturomics 2.0: Forecasting large-scale human behavior using global news media tone in time and space. *First Monday [Online]*, Vol 16 No 9, 2011.

V. Lifschitz. John McCarthy (1927-2011). *Nature*, 480(7375):40–40, Dec. 2011. doi: 10.1038/480040a. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=22129718&retmode=ref&cmd=prlinks.

J. Lin. Scalable language processing algorithms for the masses: a case study in computing word co-occurrence matrices with MapReduce. In *EMNLP '08: Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 419–428, Morristown, NJ, USA, 2008. Association for Computational Linguistics.

J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool, 2010. ISBN 978-1608453429. URL http://www.umiacs.umd.edu/~jimmylin/book.html.

A. Lith and J. Mattson. Investigating storage solutions for large data. Master's thesis, Master's thesis, Department of Computer Science and Engineering–Chalmers University of Technology, 2010. URL http://jakobadam.googlecode.com/svn-history/r161/trunk/src/msc-report.pdf.

B. Liu. Sentiment Analysis and Subjectivity. In N. Indurkhya and F. J. Damerau, editors, *Handbook of Natural Language Processing*, pages 1–38. Chapman and Hall/CRC, 2010. ISBN 978-1420085921. URL http://www.google.de/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CF8QFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.216.5533%26rep%3Drep1%26type%3Dpdf&ei=LBrHT8DHCIbhtQaOmezuDg&usg=AFQjCNGlIEx2-18_RFIfCe09Le8ScuTVrQ.

A. Loddengaard. Cloudera's Support Team Shares Some Basic Hardware Recommendations. Mar. 2010. URL http://www.cloudera.com/blog/2010/03/clouderas-support-team-shares-some-basic-hardware-recommendations/.

O. Lorenzo, R. Piqueras, J. J. Sánchez-Serrano, and R. Solano. ETHYLENE RESPONSE FACTOR1 integrates signals from ethylene and jasmonate pathways in plant defense. *The Plant cell*, 15(1):165–178, Jan. 2003. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=12509529&retmode=ref&cmd=prlinks.

T. Loughran and B. McDonald. When Is a Liability Not a Liability? Textual Analysis, Dictionaries, and 10-Ks. *The Journal of Finance*, 66(1):35–65, 2011. URL http://www.afajof.org/journal/abstract.asp?ref=0022-1082&vid=66&iid=1&aid=3&s=-9999.

Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed GraphLab: a framework for machine learning and data mining in the cloud. In *Proceedings of the VLDB Endowment*. VLDB Endowment, Apr. 2012. URL http://dl.acm.org/citation.cfm?id=2212354.

G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD '10: Proceedings of the 2010 international conference on Management of data*. ACM Request Permissions, June 2010. doi: 10.1145/1807167.1807184. URL http://portal.acm.org/citation.cfm?id=1807167.1807184&coll=DL&dl=GUIDE&CFID=105156848&CFTOKEN=50434359.

C. D. Manning and H. Schuetze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1 edition, June 1999. ISBN 0262133601. URL http://www.amazon.com/Foundations-Statistical-Natural-Language-Processing/dp/0262133601%3FSubscriptionId%3D1V7VTJ4HA4MFT9XBJ1R2%26tag%3Dmekentosjcom-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0262133601.

C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. URL http://nlp.stanford.edu/IR-book/information-retrieval-book.html.

M. Marcus, M. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL http://dl.acm.org/citation.cfm?id=972475.

L. Marquez, X. Carreras, K. C. Litkowski, and S. Stevenson. Semantic role labeling: An introduction to the special issue. *Computational Linguistics*, 34(2):145–159, 2008. URL http://www.mitpressjournals.org/doi/abs/10.1162/coli.2008.34.2.145.

S. J. Matthews and T. L. Williams. MrsRF: an efficient MapReduce algorithm for analyzing large collections of evolutionary trees. *BMC Bioinformatics*, 11 Suppl 1:S15, 2010. doi: 10.1186/1471-2105-11-S1-S15. URL http://dx.doi.org/10.1186/1471-2105-11-S1-S15.

P. Mell and T. Grance. The NIST Definition of Cloud Computing, 2009.

H.-W. Mewes, B. Wachinger, and V. Stümpflen. Perspectives of a systems biology of the synapse: How to transform an indefinite data space into a model? *Pharmacopsychiatry*, 43 Suppl 1:S2–S8, May 2010. doi: 10.1055/s-0030-1249666. URL http://dx.doi.org/10.1055/s-0030-1249666.

H.-W. Mewes, B. Wachinger, and V. Stümpflen. Mit semantischem Text Mining von biologischen Netzwerken zum Börsenkurs. *Biospektrum*, 12(3):333–335, 2012.

J.-B. Michel, Y. K. Shen, A. P. Aiden, A. Veres, M. K. Gray, T. G. B. Team, J. P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, S. Pinker, M. Nowak, and E. Lieberman-Aiden. Quantitative analysis of culture using millions of digitized books. *Science*, 331:176–182, 2011. URL http://www.sciencemag.org/content/331/6014/176.full.html.

M. Miwa, R. Saetre, J.-D. Kim, and J. Tsujii. Event extraction with complex event classification using rich features. *J Bioinform Comput Biol*, 8(1):131–146, Feb. 2010.

M. Miwa, P. Thompson, and S. Ananiadou. Boosting automatic event extraction from the literature using domain adaptation and coreference resolution. *Bioinformatics*, Apr. 2012. doi: 10.1093/bioinformatics/bts237. URL http://dx.doi.org/10.1093/bioinformatics/bts237.

Y. Miyao, T. Ohta, K. Masuda, Y. Tsuruoka, K. Yoshida, T. Ninomiya, and J. Tsujii. Semantic retrieval for the accurate identification of relational concepts in massive textbases. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 1017–1024, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. doi: 10.3115/1220175.1220303. URL http://dx.doi.org/10.3115/1220175.1220303.

F. Mosteller and D. Wallace. Inference in an authorship problem. *Journal of the American Statistical Association*, pages 275–309, 1963. URL http://www.jstor.org/stable/10.2307/2283270.

K. Nenova. *Towards Effective Biomedical Knowledge Discovery through Subject-Centric Semantic Integration of the Life-Science Information Space*. PhD thesis, Technische Universität München, Lehrstuhl für Genomorientierte Bioinformatik, 2008.

D. Noble. *The Music of Life: Biology Beyond Genes*. Oxford University Press, 2008. ISBN 9780199228362. URL http://books.google.de/books?id=Z9QaCmsSGf8C.

L. Oesper, D. Merico, R. Isserlin, and G. D. Bader. WordCloud: a Cytoscape plugin to create a visual semantic summary of networks. *Source code for biology and medicine*, 6:7, 2011. doi: 10.1186/1751-0473-6-7. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=21473782&retmode=ref&cmd=prlinks.

N. Okazaki and S. Ananiadou. Building an abbreviation dictionary using a term recognition approach. *Bioinformatics*, 22(24):3089–3095, Dec. 2006. doi: 10.1093/bioinformatics/btl534. URL http://dx.doi.org/10.1093/bioinformatics/btl534.

N. Okazaki, S. Ananiadou, and J. Tsujii. Building a High Quality Sense Inventory for Improved Abbreviation Disambiguation. *Bioinformatics*, Mar. 2010. doi: 10.1093/bioinformatics/btq129. URL http://dx.doi.org/10.1093/bioinformatics/btq129.

J. O'Neil. Doing Things with Words, Part Two: Sentence Boundary Detection, Oct. 2008. URL http://www.attivio.com/blog/57-unified-information-access/263-doing-things-with-words-part-two-

sentence-boundary-detection.html.

L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Jan. 1998. URL http://ilpubs.stanford.edu:8090/422/.

G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, June 2005. doi: 10.1038/nature03607. URL http://dx.doi.org/10.1038/nature03607.

G. Palla, I. J. Farkas, P. Pollner, I. Derényi, and T. Vicsek. Fundamental statistical features and self-similar properties of tagged networks. *New Journal of Physics*, 10(12):123026, Dec. 2008. doi: 10.1088/1367-2630/10/12/123026. URL http://stacks.iop.org/1367-2630/10/i=12/a=123026?key=crossref.c685f639548b267040e9baeaf84c1fcf.

M. Palmer, D. Gildea, and P. Kingsbury. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics*, 31(1):71–106, 2005. doi: 10.1162/0891201053630264. URL http://www.mitpressjournals.org/doi/abs/10.1162/0891201053630264.

M. Palmer, D. Gildea, and N. Xue. *Semantic Role Labeling*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool, 2010. ISBN 978-1598298314.

J. Park. *XML Topic Maps: Creating and Using Topic Maps for the Web*. Addison-Wesley Professional, 2002. URL http://www.amazon.com/XML-Topic-Maps-Creating-Paperback/dp/B0086M5GDK%3FSubscriptionId%3D1V7VTJ4HA4MFT9XBJ1R2%26tag%3Dmekentosjcom-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3DB0086M5GDK.

A. P. Paulauskas. [Multiple forms of NAD-dependent malate dehydrogenase in 3 species of trematodes of the genus Notocotylus (Trematoda: Notocotylidae)]. *Parazitologiia*, 24(6):492–498, Nov. 1990. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=2100002&retmode=ref&cmd=prlinks.

J. P. Pestian, P. Matykiewicz, M. Linn-Gust, B. South, O. Uzuner, J. Wiebe, K. B. Cohen, J. Hurdle, and C. Brew. Sentiment Analysis of Suicide Notes: A Shared Task. *Biomedical informatics insights*, 5(Suppl 1):3–16, Jan. 2012. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=22419877&retmode=ref&cmd=prlinks.

E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure trends in a large disk drive population. In *FAST '07: Proceedings of the 5th USENIX conference on File and Storage Technologies*. USENIX Association, Feb. 2007. URL http://portal.acm.org/citation.cfm?id=1267903.1267905&coll=DL&dl=GUIDE&CFID=92133316&CFTOKEN=54312161.

Powerset. First Release of Hbase in Hadoop, Oct. 2007. URL http://www.bing.com/community/site_blogs/b/powerset/archive/2007/10/16/first-release-of-hbase-in-hadoop.aspx.

Powerset. Microsoft to Acquire Powerset, July 2008. URL http://www.bing.com/community/site_blogs/b/powerset/archive/2008/07/01/microsoft-to-acquire-powerset.aspx.

S. Pradhan and W. Ward. Towards robust semantic role labeling. *Computational Linguistics*, 2008. URL http://www.mitpressjournals.org/doi/abs/10.1162/coli.2008.34.2.289.

S. R. Proulx, D. E. L. Promislow, and P. C. Phillips. Network thinking in ecology and evolution. *Trends in ecology & evolution*, 20(6):345–353, June 2005. doi: 10.1016/j.tree.2005.04.004. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=16701391&retmode=ref&cmd=prlinks.

V. Punyakanok, D. Roth, and W. Yih. The Importance of Syntactic Parsing and Inference in Semantic Role Labeling. *Computational Linguistics*, 34(2), 2008. URL http://cogcomp.cs.illinois.edu/papers/PunyakanokRoYi07.pdf.

S. Pyysalo, F. Ginter, J. Heimonen, J. Björne, J. Boberg, J. Järvinen, and T. Salakoski. BioInfer: a corpus for information extraction in the biomedical domain. *BMC Bioinformatics*, 8:50, 2007. doi: 10.1186/1471-2105-8-50. URL http://dx.doi.org/10.1186/1471-2105-8-50.

A. Radwan. MapReduce 2.0 in Hadoop 0.23, Feb. 2012. URL http://www.cloudera.com/blog/2012/02/mapreduce-2-0-in-hadoop-0-23/.

D. Rebholz-Schuhmann, H. Kirsch, and F. Couto. Facts from text–is text mining ready to deliver? *PLoS Biol*, 3(2):e65, Feb. 2005. doi: 10.1371/journal.pbio.0030065. URL http://dx.doi.org/10.1371/journal.pbio.0030065.

J. C. Reynar and A. Ratnaparkhi. A Maximum Entropy Approach to Identifying Sentence Boundaries. *arXiv.org*, cmp-lg, Apr. 1997. URL http://arxiv.org/abs/cmp-lg/9704002v1.

L. Richardson, S. Ruby, and D. H. Hansson. *Restful Web Services*. O'Reilly Media, first edition edition, May 2007. ISBN 0596529260. URL http://www.amazon.com/Restful-Web-Services-Leonard-Richardson/dp/0596529260%3FSubscriptionId%3D1V7VTJ4HA4MFT9XBJ1R2%26tag%3Dmekentosjcom-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0596529260.

K. Rohloff and R. E. Schantz. Clause-Iteration with MapReduce to Scalably Query Data Graphs in the SHARD Graph-Store. In *DIDC 11*, 2011.

M. Roth, T. Barenholz, A. Ben-David, D. Deutscher, G. Flysher, A. Hassidim, I. Horn, A. Leichtberg, N. Leiser, Y. Matias, and R. Merom. Suggesting (More) Friends Using the Implicit Social Graph. In *International Conference on Machine Learning (ICML)*, 2011.

A. Rzhetsky, M. Seringhaus, and M. B. Gerstein. Getting started in text mining: part two. *PLoS Comput Biol*, 5(7):e1000411, July 2009. doi: 10.1371/journal.pcbi.1000411. URL http://dx.doi.org/10.1371/journal.pcbi.1000411.

M. Sahami, S. Dumais, D. Heckerman, and E. Horovitz. A Bayesian Approach to Filtering Junk E-Mail. In *AAAI'98 Workshop on Learning for Text Categorization*. ACM Request Permissions, 1998. URL http://portal.acm.org/citation.cfm?id=1953163.1953245&coll=DL&dl=GUIDE&CFID=138408822&CFTOKEN=52827053.

A. Sarje and S. Aluru. A MapReduce Style Framework for Computations on Trees. In *ICPP '10: Proceedings of the 2010 39th International Conference on Parallel Processing*. IEEE Computer Society, Sept. 2010. URL http://portal.acm.org/citation.cfm?id=1904934.1905224&coll=DL&dl=GUIDE&CFID=100082333&CFTOKEN=90081878.

A. S. Schwartz and M. A. Hearst. A simple algorithm for identifying abbreviation definitions in biomedical text. *Pac Symp Biocomput*, pages 451–462, 2003.

D. B. Searls. The language of genes. *Nature*, 420(6912):211–217, Nov. 2002. doi: 10.1038/nature01255. URL http://dx.doi.org/10.1038/nature01255.

S. Shankland. We're all guinea pigs in Google's search experiment, May 2008a. URL http://news.cnet.com/8301-10784_3-9954972-7.html?tag=mncol;txt.

S. Shankland. Google spotlights data center inner workings, May 2008b. URL http://news.cnet.com/8301-10784_3-9955184-7.html.

J.-H. Shin, V. L. Dawson, and T. M. Dawson. SnapShot: pathogenesis of Parkinson's disease. *Cell*, 139(2): 440.e1–2, Oct. 2009. doi: 10.1016/j.cell.2009.09.026. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=19837042&retmode=ref&cmd=prlinks.

Sigma-Aldrich. Hematopoietic Stem Cells. URL http://www.sigmaaldrich.com/life-science/stem-cell-biology/hematopoietic-stem.html.

P. Smialowski, P. Pagel, P. Wong, B. Brauner, I. Dunger, G. Fobo, G. Frishman, C. Montrone, T. Rattei, D. Frishman, and A. Ruepp. The Negatome database: a reference set of non-interacting protein pairs. *Nucleic Acids Res*, 38(Database issue):D540–D544, 2010. doi: 10.1093/nar/gkp1026. URL http://nar.oxfordjournals.org/content/38/suppl_1/D540.long.

W. W. Smith, Z. Pei, H. Jiang, V. L. Dawson, T. M. Dawson, and C. A. Ross. Kinase activity of mutant LRRK2 mediates neuronal toxicity. *Nature Neuroscience*, 9(10):1231–1233, Sept. 2006. doi: 10.1038/nn1776. URL http://www.nature.com/doifinder/10.1038/nn1776.

A. Stevenson and C. A. Lindberg. *New Oxford American Dictionary*. Oxford University Press, 3rd edition edition, Sept. 2010. ISBN 978-0195392883. URL http://www.amazon.de/Oxford-American-Dictionary-With-CDROM/dp/0195170776.

R. Strache. Institute of Bioinformatics and Systems Biology Helmholtz Center MunichMaster Thesis. Master's thesis, Jan. 2012.

V. Stümpflen and B. Wachinger. Datenschätze heben: Systembiologie mit dem Semantic Web. *GenomXPress*, 4.09:4–6, 2009.

S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. *Proceedings of the 20th international conference on World wide web*, pages 607–614, 2011. URL http://dl.acm.org/citation.cfm?id=1963491.

D. R. Swanson. Fish oil, Raynaud's syndrome, and undiscovered public knowledge. *Perspect Biol Med*, 30(1): 7–18, 1986.

J. G. P. Tarasewich. Improved text entry for mobile devices: alternate keypad designs and novel predictive disambiguation methods. *Improved text entry for mobile devices: alternate keypad designs and novel predictive disambiguation methods*, Jan. 2007. URL http://portal.acm.org/citation.cfm?id=1415127&coll=DL&dl=GUIDE&CFID=147238308&CFTOKEN=42836571.

A. Taylor. Cypher - A view from a recovering SQL DBA. Dec. 2011. URL http://systay.github.com/blog/2011/11/06/cypher---a-view-from-a-recovering-sql-dba/.

R. C. Taylor. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinformatics*, 11 Suppl 12:S1, 2010. doi: 10.1186/1471-2105-11-S12-S1. URL http://dx.doi.org/10.1186/1471-2105-11-S12-S1.

The Neo4j Team. *The Neo4j Manual*, 1.8.m01 edition, May 2012. URL http://docs.neo4j.org/pdf/neo4j-manual-milestone.pdf.

A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy. Hive-a petabyte scale data warehouse using hadoop. *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 996–1005, 2010. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5447738.

K. Toutanova and C. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In: Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora(EMNLPNLC-2000). *Hong Kong, China.*, 2000. URL http://scholar.google.com/scholar?q=related:30LXm3HUgFYJ:scholar.google.com/&hl=en&num=30&as_sdt=0,5.

K. Toutanova, D. Klein, C. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180, 2003. URL http://dl.acm.org/citation.cfm?id=1073478.

K. Toutanova, A. Haghighi, and C. D. Manning. A global joint model for semantic role labeling. *Computational Linguistics*, 34(2), June 2008. URL http://www.mitpressjournals.org/doi/abs/10.1162/coli.2008.34.2.161.

J. Travers and S. Milgram. An experimental study of the small world problem. *Sociometry*, pages 425–443, 1969. URL http://www.jstor.org/stable/10.2307/2786545.

O. Trelles, P. Prins, M. Snir, and R. C. Jansen. Big data, but are we ready? *Nat Rev Genet*, 12(3):224, Mar. 2011. doi: 10.1038/nrg2857-c1. URL http://dx.doi.org/10.1038/nrg2857-c1.

R. T.-H. Tsai, W.-C. Chou, Y.-S. Su, Y.-C. Lin, C.-L. Sung, H.-J. Dai, I. T.-H. Yeh, W. Ku, T.-Y. Sung, and W.-L. Hsu. BIOSMILE: a semantic role labeling system for biomedical verbs using a maximum-entropy model with automatically generated template features. *BMC Bioinformatics*, 8:325, 2007. doi: 10.1186/1471-2105-8-325. URL http://dx.doi.org/10.1186/1471-2105-8-325.

A. M. Turing. Computing Machinery and Intelligence. pages 1–21, 1950.

J. H. G. M. van Beek. Channeling the Data Flood: Handling Large-Scale Biomolecular Measurements in Silico. In *Proceedings of the IEEE*, Apr. 2006.

A. Verma, X. Llorà, R. H. Campbell, and D. E. Goldberg. Scaling Genetic Algorithms using MapReduce. *IlliGAL Report*, No. 2009007, 2009.

B. Wachinger. Genesis and Effects of LRRK2 in Parkinson's Disease: Inference of a Qualitative Model from Distributed Knowledge. Master's thesis, June 2009.

B. Wachinger and V. Stümpflen. Scaling Biomedical Topic Maps to Billions of Associations: How to Cope With Terabytes of Data? In *TMRA 2010*, Sept. 2010.

B. Wachinger, S. T. Haque, U. Köhler, S. Ullherr, and V. Stümpflen. Excerbt: Next-Generation Knowledge Extraction and Hypothesis Generation from Massive Amounts of Biomedical Literature. In *ICSB 2012*, Aug.

2012.

M. C. Walter, I. Dunger, B. Wachinger, and D. Frangoulidis. Knowledge Digging in Coxiella Burnetii - Bioinformatics Tools for the 21st Century Science. In *Biodefense*, 2011.

T. Wattarujeekrit, P. K. Shah, and N. Collier. PASBio: predicate-argument structures for event extraction in molecular biology. *BMC Bioinformatics*, 5:155, Oct. 2004. doi: 10.1186/1471-2105-5-155. URL http://dx.doi.org/10.1186/1471-2105-5-155.

D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998. doi: 10.1038/30918. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=9623998&retmode=ref&cmd=prlinks.

J. Webber. Graph Processing versus Graph Databases. 2011. URL http://jim.webber.name/2011/08/24/66f1fb4b-83c3-4f52-af40-ee6382ad2155.aspx.

E. W. Weisstein. Wolfram MathWorld, 1999. URL http://mathworld.wolfram.com/.

J. Weizenbaum. ELIZA - a computer program for the study of natural language communication between man and machine. *Commun. ACM ()*, 9(1):36–45, 1966. doi: 10.1145/365153.365168. URL http://doi.acm.org/10.1145/365153.365168.

J. Weston, R. Kuang, C. Leslie, and W. S. Noble. Protein ranking by semi-supervised network propagation. *BMC Bioinformatics*, 7 Suppl 1:S10, 2006. doi: 10.1186/1471-2105-7-S1-S10. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=16723003&retmode=ref&cmd=prlinks.

T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, original edition, June 2009. ISBN 0596521979. URL http://www.amazon.com/Hadoop-Definitive-Guide-Tom-White/dp/0596521979%3FSubscriptionId%3D1V7VTJ4HA4MFT9XBJ1R2%26tag%3Dmekentosjcom-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0596521979.

A. Wilson and A. Trumpp. Bone-marrow haematopoietic-stem-cell niches. *Nature reviews. Immunology*, 6 (2):93–106, Feb. 2006. doi: 10.1038/nri1779. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=16491134&retmode=ref&cmd=prlinks.

A. Winkler. Semantic Named Entity Type Disambiguation via Bayesian Inference. Master's thesis, Ludwig-Maximilians-University and Technical University Munich, Helmholtz Zentrum München, 2011.

J. D. Wren. Extending the mutual information measure to rank inferred literature relationships. *BMC Bioinformatics*, 5:145, Oct. 2004. doi: 10.1186/1471-2105-5-145. URL http://dx.doi.org/10.1186/1471-2105-5-145.

A. Wright. Mining the Web for Feelings, Not Facts, Aug. 2009. URL http://www.nytimes.com/2009/08/24/technology/internet/24emotion.html?_r=2.

J. Xiang. HBase Write Path, June 2012. URL http://www.cloudera.com/blog/2012/06/hbase-write-path/.

Yahoo. The History of Yahoo! - How It All Started..., 2005. URL http://docs.yahoo.com/info/misc/history.html.

H. Yu, P. Braun, M. A. Yildirim, I. Lemmens, K. Venkatesan, J. Sahalie, T. Hirozane-Kishikawa, F. Gebreab, N. Li, N. Simonis, T. Hao, J. F. Rual, A. Dricot, A. Vazquez, R. R. Murray, C. Simon, L. Tardivo, S. Tam, N. Svrzikapa,

C. Fan, A. S. de Smet, A. Motyl, M. E. Hudson, J. Park, X. Xin, M. E. Cusick, T. Moore, C. Boone, M. Snyder, F. P. Roth, A. L. Barabasi, J. Tavernier, D. E. Hill, and M. Vidal. High-Quality Binary Protein Interaction Map of the Yeast Interactome Network. *Science*, 322(5898):104–110, Oct. 2008. doi: 10.1126/science.1158684. URL http://www.sciencemag.org/cgi/doi/10.1126/science.1158684.

S. Zhang, X. Ning, and X.-S. Zhang. Identification of functional modules in a PPI network by clique percolation clustering. *Computational Biology and Chemistry*, 30(6):445–451, Dec. 2006. doi: 10.1016/j.compbiolchem.2006.10.001. URL http://linkinghub.elsevier.com/retrieve/pii/S1476927106000867.

W. Zhao, H. Ma, and Q. He. Parallel K-Means Clustering Based on MapReduce. *Lecture Notes in Computer Science: Cloud Computing*, pages 674–679, 2009. doi: 10.1007/978-3-642-10665-1{_}71.

G. K. Zipf. *Selected Studies of the Principle of Relative Frequency in Language*. Harvard University Press, first edition, 1932.

J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6, 2006. doi: http://doi.acm.org/10.1145/1132956.1132959.

Publication Record

At the time of submission of this thesis, the following publications (journal articles and conference proceedings) with my participation and an association to the work presented in this thesis have been published, submitted, or were in preparation (in reverse publication order):

[1] **B. Wachinger**, U. Köhler, S. Ullherr, R. Strache, T. Barnickel, H.W. Mewes, V. Stümpflen. Excerbt: A Next-Generation Text Mining Engine for Qualitative Modeling in the Big Data Age. 2012. *in preparation*.

[2] N. Gross, P. Blohm, **B. Wachinger**, T. Barnickel, H.W. Mewes, V. Stümpflen. DefineTHAT – Automated Literature Based Extraction of Definitions for Biomedical Terms. 2012. *in preparation*.

[3] M. Arnold, M.L. Hartsperger, E. Rodríguez, H. Baurecht, **B. Wachinger**, A. Franke, M. Kabesch, J. Winkelmann, M. Romanos, T. Illig, H.W. Mewes, V. Stümpflen, S. Weidinger. Network-based data prioritization approach identifies joint and disjoint genetic features across common human diseases. BMC Genomics 2012, 13:490. [citation: Arnold et al. (2012)].

[4] **B. Wachinger**, S. T. Haque, U. Köhler, S. Ullherr, V. Stümpflen. Excerbt: Next-Generation Knowledge Extraction from Massive Amounts of Biomedical Literature. 13th International Conference on Systems Biology (ICSB), August 2012, Toronto, Canada. [citation: Wachinger et al. (2012)]

[5] H.W. Mewes, **B. Wachinger**, and V. Stümpflen. Web Intelligence: Mit semantischem Text Mining von biologischen Netzwerken zum Börsenkurs. BIOspektrum 03.12, 2012. [citation: Mewes et al. (2012)]

[6] M.C. Walter, I. Dunger, **B. Wachinger**, D. Frangoulidis. Knowledge Digging in Coxiella Burnetii – Bioinformatics Tools for the 21st Century Science. Biodefense, 2011. [citation: Walter et al. (2011)]

[7] **B. Wachinger** and V. Stümpflen. Scaling Biomedical Topic Maps to Billions of Associations: How to Cope With Terabytes of Data? Sixth International Conference on Topic Maps Research and Applications (TMRA), October 2010, Leipzig, Germany. [citation: Wachinger and Stümpflen (2010)]

[8] H.W. Mewes, **B. Wachinger**, and V. Stümpflen. Perspectives of a Systems Biology of the Synapse: How to Transform an Indefinite Data Space into a Model? Pharmacopsychiatry, 43 Suppl 1:S2–S8, May 2010. [citation: Mewes et al. (2010)]

[9] V. Stümpflen and **B. Wachinger**. Datenschätze heben: Systembiologie mit dem Semantic Web. GenomXPress, 2009. [citation: Stümpflen and Wachinger (2009)]

[10] **B. Wachinger**. Genesis and Effects of LRRK2 in Parkinson's Disease: Inference of a Qualitative Model from Distributed Knowledge. Diploma thesis, 2009. [citation: Wachinger (2009)]

Declaration / Erklärung

Ich erkläre an Eides statt, dass ich die bei der promotionsführenden Einrichtung Fakultät Wissenschaftszentrum Weihenstephan für Ernährung, Landnutzung und Umwelt der Technischen Universität München zur Promotionsprüfung vorgelegte Arbeit mit dem Titel

Next Generation Knowledge Extraction from Biomedical Literature with Semantic Big Data Approaches

am Institut für Bioinformatik und Systembiologie des Helmholtz Zentrum München unter der Anleitung und Betreuung durch *Prof. Dr. Hans-Werner Mewes* ohne sonstige Hilfe erstellt und bei der Abfassung nur die gemäß § 6 Abs. 6 und 7 Satz 2 angegebenen Hilfsmittel benutzt habe.

- ☒ Ich habe keine Organisation eingeschaltet, die gegen Entgelt Betreuerinnen und Betreuer für die Anfertigung von Dissertationen sucht, oder die mir obliegenden Pflichten hinsichtlich der Prüfungsleistungen für mich ganz oder teilweise erledigt.
- ☒ Ich habe die Dissertation in dieser oder ähnlicher Form in keinem anderen Prüfungsverfahren als Prüfungsleistung vorgelegt.
- □ Die vollständige Dissertation wurde in ..... veröffentlicht. Die promotionsführende Einrichtung ..... hat der Vorveröffentlichung zugestimmt.
- ☒ Ich habe den angestrebten Doktorgrad noch nicht erworben und bin nicht in einem früheren Promotionsverfahren für den angestrebten Doktorgrad endgültig gescheitert.
- □ Ich habe bereits am ..... bei der Fakultät für ..... der Hochschule ..... unter Vorlage einer Dissertation mit dem Thema die Zulassung zur Promotion beantragt mit dem Ergebnis: ..... .

Die öffentlich zugängliche Promotionsordnung der Technischen Universität München ist mir bekannt, insbesondere habe ich die Bedeutung von §28 (Nichtigkeit der Promotion) und §29 (Entzug des Doktorgrades) zur Kenntnis genommen. Ich bin mir der Konsequenzen einer falschen Eidesstattlichen Erklärung bewusst.

Mit der Aufnahme meiner personenbezogenen Daten in die Alumni-Datei bei der Technischen Universtität München bin ich

- ☒ einverstanden
- □ nicht einverstanden

München, den .....

.....
Benedikt Wachinger