Sönke Lieberam-Schmidt

Analyzing and Influencing Search Engine Results

Business and Technology Impacts on Web Information Retrieval

RESEARCH

Sönke Lieberam-Schmidt

Analyzing and Influencing Search Engine Results

GABLER RESEARCH

Sönke Lieberam-Schmidt

Analyzing and Influencing Search Engine Results

Business and Technology Impacts on Web Information Retrieval

With a foreword by Prof. Dr. Erwin Pesch

GABLER

RESEARCH

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Doctoral thesis, University of Siegen, 2009

1st Edition 2010

Editorial Office: Ute Wrasmann | Sabine Schöller

Gabler Verlag is a brand of Springer Fachmedien.
Springer Fachmedien is part of Springer Science+Business Media.
www.gabler.de

Cover design: KünkelLopka Medienentwicklung, Heidelberg
Printed on acid-free paper
Printed in Germany

ISBN 978-3-8349-2453-7

Foreword

Search engines are indexing several billions of webpages and thus enable an extremely quick access to all sorts of contents of the Web. Most of these requests consist of one word. As result, several thousands of documents are delivered where the first documents are not necessarily the needed ones. Where the specification of the query depends on the searcher's knowledge, the order of listed results depends on the search engine used. Knowing the search engines leads to a better understanding of results and helps developing specific Web-based sales strategies.

The present work "Analyzing and Influencing Search Engine Results" picks up an up-to-date topic due to the fact that search engines have become an important influence factor in economy and society.

The author presents a comprehensive overview of recent Web-search technology. He describes the related problems from a search engine's as well as from the user's perspective. Driven by the users' needs, new approaches for searching the Web are developed, implemented and tested.

Readers of this work will become familiar with the precautions that have to be taken for reaching upper result page positions. Different strategies are explained including spamming techniques that are used to influence the results of ranking methods.

Following the principle of business informatics, the author explores the topic from a business and a technology view. He complements the scientific view by approaches for practical problems and gives concrete instructions for practitioners. The work delivers a deep insight into the methods and problems of Web search and is well suited for Website owners that want to improve their visibility in the Web.

Seeing the high relevance of search engines in many different areas, I expect this work to attract high attention of the scientific community and the practice.

Professor Dr. Erwin Pesch

Acknowledgements

During the development of this work, several people and organizations supported me exploring the world of Web search. I strongly appreciate their contribution and I want to express my thanks to all of them. Nevertheless, I like to name some of them in the following.

First, I want to thank Prof. Dr. Erwin Pesch and his team of the chair of management information sciences for supplying the research environment, their input in stimulating discussions and their constructive feedback. Prof. Dr. Herbert Kopfer and Prof. Dr. Joachim Eigler receive my thanks for conducting a smooth and fast review and examination process. I thank my colleagues and the staff of Siegen University for building a productive and convenient working atmosphere.

Furthermore, I want to thank the contraco Consulting & Software Ltd., Potsdam, for their contribution to the funding of this work and the various inspirations I have received from their management and staff. Different co-operations with Vivawasser AG, Kiel, have also resulted into productive input that I gratefully acknowledge.

The companies T-Online International AG, Darmstadt, and FAST Search & Transfer ASA, Oslo, gave me useful insights into the details and practical problems of search engine operations. I thank the Interessengemeinschaft Frankfurter Kreditinstitute GmbH who expressed their appreciation for this work by funding its publication.

Last but not least, I wish to express my deepest gratitude to my family and my friends – and especially my fiancée – for the strong support they have given me during the time I have been working on this project.

Dr. Sönke Lieberam-Schmidt

Contents

List of Figures

List of Tables

List of Abbreviations

ALA	Aroundlink Algorithm
ASP	Active Server Pages
CERN	European Organization for Nuclear Research, originally Conseil Européen pour la Recherche Nucléaire
DFS	Distributed File System
DNS	Domain Name System
EKSPM	Enhanced Keyword Selection Position Model
EM	Expectation Maximization
GB	Gigabyte(s)
GUI	Graphical User Interface
HITS	Hyperlink Induced Topic Search
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
ID	Identification
idf	Inverse Document Frequency
IP	Internet Protocol
ISE	Information Supply Engine
JSP	JavaServer Pages
JVM	Java Virtual Machine

kB	Kilobyte(s)
KSPM	Keyword Selection Position Model
KSTPM	Keyword Selection Top Position Model
LSI	Latent Semantic Indexing
MB	Megabyte(s)
ms	Millisecond(s)
ODP	Open Directory Project
PC	Personal Computer
PHP	recursive initialism for PHP: Hypertext Preprocessor
PLSI	Probabilistic Latent Semantic Indexing
ROCK	RObust Clustering using linKs
s	Second(s)
SCC	Strongly connected component
SEO	Search Engine Optimization
SERP	Search-Engine Result-Page
SHOC	Semantic Hierarchical Online Clustering
SVD	Singular Value Decomposition
TEM	Tempered Expectation Maximization
tf	Term Frequency
U.S.	United States
URL	Uniform Resource Locator
VPA	Vox Populi Algorithm
VSM	Vector Space Model
XHTML	Extensible Hypertext Markup Language

Introduction

The World Wide Web can be compared with a giant warehouse containing millions of products that are stored in shelves without any order. In this image, socks would lie next to sausages, and soap maybe in the freezer near the newspapers. Even worse, many products could not be found at all because they are buried underneath heaps of other products. Besides the domain name system (DNS) which regulates Internet addresses, there is not any helpful intrinsic structure in the Web.

Search engines can assist in navigating through the Web. Many Internet users employ them as an entry point into the World Wide Web. But often, search engines yield an overwhelming and confusing amount of results for a search query. Thus, the order of webpages presented to the user has a major impact on which offers of the Web are used. The quality of a search engine cannot only be evaluated with respect to the quantity of obtained search results. This might have been the case at the beginning of Web search several years ago. Besides the result quality, determined by e.g. finding the right answer for a search query or the absence of irrelevant results, the ability to present search results in a clear manner has become a crucial point of interest.

This work gives an overview of current Web search techniques and delivers new solutions for search related problems to the three parties involved in the search process: *Web search users, search engine operators,* and *website providers*.

Overview

The first chapter *(Web Search)* illustrates the importance of Web search showing the development of Internet usage as well as search engine usage. The existing information demand is represented by the user's information need and the user behavior. After describing search tools used to fulfill this demand, current methods used by search engines to deliver results are explained. A new approach for measuring relevancy is presented. The chapter closes with the evolution history of search engines and an outlook to possible future search engines.

Chapter 2 *(Web Structure)* deals with the World Wide Web's role as an information supplier. After an analysis of the Web's contents and its graph structure, different methods are presented that benefit from the characteristics of the Web's structure. Own data structures for storing Web search results are presented. Their

practical relevance is shown by their implementation and test for the use of real-life data.

In Chapter 3 *(Result Clustering)*, methods for improving results gained from a search engine are developed. Different methods for clustering search engine results are implemented and tested. They are evaluated based on practical tests on real data.

Chapter 4 *(Search Engine Optimization, SEO)* analyzes the impact of search techniques for website owners and shows how they can use these techniques to improve their sites' visibility in the World Wide Web. After analyzing the objectives of website owners, the possibilities and means of search engine optimization are explained. New models for keyword selection are introduced as a basis for optimization efforts. They are implemented in the framework of an SEO process that is developed in co-operation with a Web service provider. The success of the process implementation is proved on two different real-life projects.

The scientific contributions of this work and their practical applications are summarized and complemented by opportunities for further work in Chapter 5 *(Conclusion).*

Explanation of Basic Terms

For a common understanding, some basic terms used in this work are explained in the following.

A *webpage* is a document made available on the World Wide Web. It is typically accessed via the Internet through a Web browser and displayed on a computer. A webpage is usually written in plain text with formatting instructions of Hypertext Markup Language (HTML, XHTML). Descriptions of the webpage that will not be displayed can be entered in *metatags*. A webpage may provide navigation functionality to other webpages via hypertext links. A webpage may also incorporate elements from other webpages.

A webpage can be accessed using a *uniform resource locator (URL)* that is e.g. typed in the address field of a Web browser. The URL specifies where the resources are available on the World Wide Web and the technical mechanism for retrieving it. It is also referred to as *Web address*.

A *hypertext link* (often referred to as *hyperlink* or *link*) on a webpage is a reference to another document or piece of information. The part of the webpage that contains a hyperlink is called *anchor* (or *anchor text* if it consists of words). The referenced piece of information is called *target* and is usually specified by the target's URL. With other words, the hyperlink on a webpage *points to* the target. Using webpages as vertices and hyperlinks as edges, a *Web graph* can be constructed.

A *website* is a collection of related webpages (or images, videos or other digital assets) under a common ownership that are addressed with a common *domain name* (e.g. `youtube.com`) or *IP address* (e.g. `141.99.1.1`). It is hosted on at least one *Web server (host)* that can be identified by its *hostname*. The Web server usually writes data of all events of document access in a *log file*, e.g. the requested URL, the time, the referring URL and some information about the user's technical equipment.

Search engines are tools that are designed to retrieve information on the World Wide Web. The input of a search engine usually is a *query* consisting of one or more *keywords* building a *query term*. The output can be webpages, images and other types of files. The results are presented on a *search-engine result-page (SERP)* and are called *hits*. The SERP lists hyperlinks pointing to the information found together with a describing text *(snippet* or *teaser)*. The database a search engine uses to store webpage information is usually referred to as *index*.

1 Web Search

Among the most used top-ten Internet activities of Internet users, search engine usage holds position two after email usage (Fallows, 2008; PEW, 2009). More than sixty percent of U.S. Internet users are using search engines on an average day, compared to eighty percent of the users who are sending or receiving emails on a typical day. These activities are followed by looking for news and job-related research (Rainie and Shermak, 2005). Table 1.1 shows the complete list of activities combined with the proportion of U.S. Internet users. For many of these activities like getting news or job-related research, search engines are the starting point to enter the World Wide Web.

The PEW Internet & American Life Project regularly examines people's Internet-usage habits. Figure 1.1 shows the percentage of the American population using email and search engines on an average day; see Rainie and Shermak (2005) and PEW (2007). Looking at the whole American population (adults of the age 18 and older), search engine usage has increased from about thirty percent in 2002 up to almost forty-five percent in 2006. Note that the basis for the percentage in Figure 1.1 is the whole population of the U.S., whereas Table 1.1 regards only Internet

Internet activities	Proportion of Internet users
Email	80 %
Search engine	*63 %*
Get news	46 %
Do job-related research	29 %
Use instant messaging	18 %
Do online banking	18 %
Take part in chat room	8 %
Make a travel reservation	5 %
Read blogs	3 %
Participate in online auction	3 %

Table 1.1: Internet activities of U.S. Internet users on a typical online day (Rainie and Shermak, 2005)

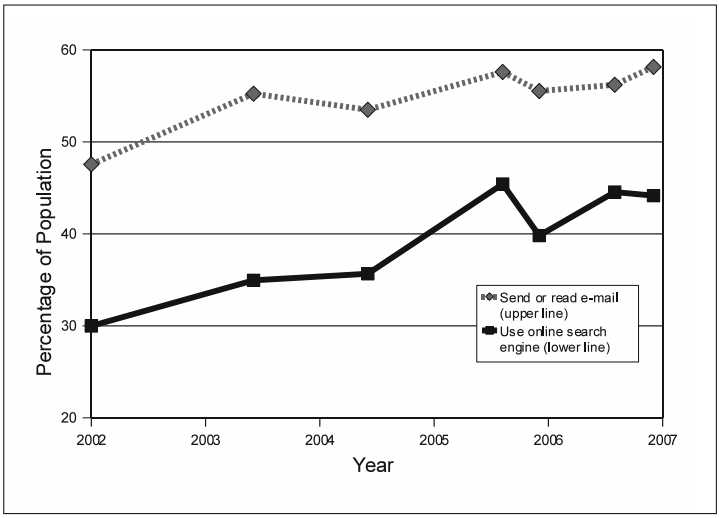


Figure 1.1: Email and search engine usage of the U.S. population on an average day

users. In the same time, the leading activity email has only increased from 48 % to 58 %. This means that search engine usage is approaching email as a primary Internet activity.

Asking Internet users if they have ever performed certain Internet activities, there is hardly a difference between the size of email-using population and the size of search engine using population. The survey shows that 91 % of all Internet users had ever sent or received email and 90 % of Internet users had ever used search engines.

Another PEW Report (Fallows, 2005) states that one third of surveyed Internet searchers "can't live without search engines". As the Internet usage itself, the search engine usage becomes more and more part of people's daily life and influences their decision making process in every area of life. This trend is supported by recent surveys of Internet usage PEW (2009).

As a result of the increasing relevancy for consumption decisions, distributing and sales companies are focusing on potential buyers coming from search engines. They try to populate the search-engine result-list using one or more of the following three major channels:

1. *Paid advertisement*: Advertisements appearing appropriately labeled on the top or the upper right of the result page.

2. *Search engine optimization*: Using methods that are approved by search engines in order to make the pages more readable and to be found under appropriate keywords at a favorable position (see Chapter 4).

3. *Search engine spamming*: Using methods that contradict the search engine rules, e.g. in order to appear on a top position for a keyword that does not necessarily correspond to the website's content (see Subsection 4.4.1).

A sharp distinction between the latter two methods is difficult to make, because it is the user who individually decides, whether the optimization effects valuable results or spam. This and more will be discussed in the next section.

1.1 Information Demand

One of the first things people try when they start using the Internet is using a search engine. Most users quickly feel comfortable with the act of searching (Fallows, 2005). They are confident in the search results they achieve as well as in their searching capabilities. The majority of users think that search engines are a fair and unbiased source of information. How neutral the selection of search results and – what is even more important – how neutral their sequence can be, this will be discussed in Section 1.3. Anyway, Internet users are very positive about their online search experiences.

Beyond the technical part of a search engine, which basically produces a certain output for a given input, it is essential to understand why users perform searches on search engines. What users want and what they get will be presented in the next section. Search engines reach to fulfill an existing information demand of the user (Subsection 1.1.1) with an information supply (Section 2.1) existing on the Web.

1.1.1 User's Information Need

The motivation of using a search engine results from various aspects. In the theory of Information Retrieval (IR), originating from library sciences, the search process starts with a so-called *information need*. The user is seeking for some information and formulates a query in a specific query language. A system then returns those documents from a document collection (corpus) that match the query.

In lots of cases, a search inquiry leads to a very large amount of results. The user then faces the challenge of finding those results that are most relevant for him or her out of a number of sometimes several dozen result pages.

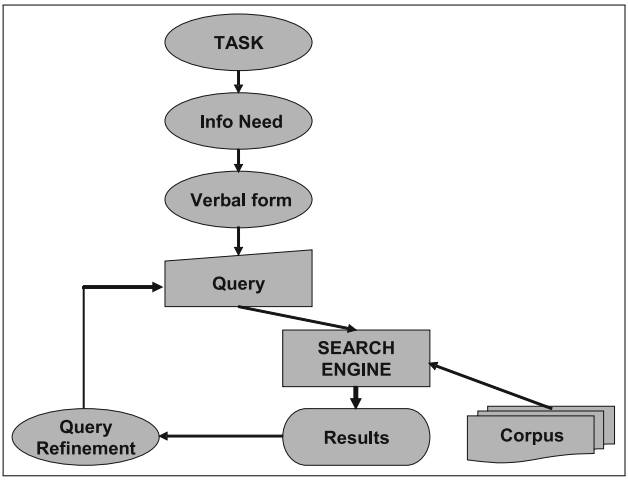


Figure 1.2: Classical search model adapted to Web search (Broder, 2002b)

In the world of the Web, besides the information need some further *need behind the query* can trigger a search process. Whereas the user may in most cases be aware of his or her need behind the query, it is almost impossible for the search engine to "know" about it. Anyway, the more precise the knowledge about the need, the better the delivered results can be. Broder (2002b) looks first at the *task* a user wants to have accomplished by the query. What kind of tasks this typically can be will be discussed later in this section. Figure 1.2 shows the classical search model adapted to Web search. A task generates an information need, which the user formulates towards a verbal form. He picks some keywords from the verbal form and creates a query to send to the search engine. If the results returned from the existing corpus do not match the user's expectations, he typically specifies the query in a query-refinement step and sends the refined query to the search engine.

The search model will be explained using a short example: A bank employee is assigned to the task "Create a Dow Jones stock portfolio". Thus, the employee needs information about shares in the Dow Jones Industrial Average index. This information need can be verbally formulated into "Which shares are contained in the Dow Jones index?". A query could be "Dow Jones", which probably leads to the Dow Jones Company and some different indexes. A suitable query refinement is "Dow Jones index" or "Dow Jones Industrial Average Index", which leads to a website with information about the composition of the index. The user can now use the information gathered to accomplish his initial task.

On the sometimes long way from the original task to suitable results different pitfalls may appear:

1. A *misconception* can lead to the situation that information thought as necessary is not what is really needed for the specific task.

2. A *mistranslation* is problematic in case the verbalization does not or not completely reflect the information need.

3. A *misformulation* of the query is the case if its formulation does not correctly reflect the verbal form of the information need.

4. *Polysemy* (i.e. one word has multiple meanings) leads to search results from a different topic which do not fit the information need.

5. *Synonymy* (i.e. the same concept can be expressed by different words) in the query term can prevent some results matching the information need from being found by the search engine.

Several approaches aim to help users overcoming these pitfalls. These approaches are part of the newest generation of search engines or of forthcoming search engines. Both will be described in detail in Section 1.5. All of the approaches have in common that they want to help the users to perform their tasks or to reach their goals, respectively.

One problem of classic search engines is that they cannot be aware of the user's goal. In the simplest case, the only thing the search engine "knows" about the user is his query consisting of a few words. Understanding the goals may become an important factor for future search interfaces. Broder (2002a,b) has classified typical goals in the following four areas:

1. *Information:* the user wants to learn or to get to know something.

2. *Navigation:* the user wants to be directed to a certain webpage.

3. *Transaction:* the user wants to carry out something using the Web.

4. *Combinations* of the former mentioned points.

In the first case, namely the *informational* query, the user assumes the information to be present on one or more webpages. He does not want to perform any further interaction except reading. In other words, the information is available in a somewhat static form. In this context, the term "static" does not refer to the

technical method of Web content creation, e.g. static HTML-pages versus dynamically created PHP-pages. It rather means that the query itself does not trigger the generation of content. The content was already there before the query process.

When the user poses a *navigational* query, he already knows about the existence of the webpage searched for, or at least he assumes that this webpage exists. This kind of query has usually one "correct" result, apart from the existence of syntactic or semantic aliases. A good link collection in the result list containing the target page may in this case also be acceptable for the user. A kind of navigational search was implemented by the search engine GOOGLE in terms of the "I'm feeling Lucky" button.[1]

The goal of *transactional* queries is to reach a website where further interaction can be conducted. This interaction can be any Web-mediated activity, e.g. online purchases, downloads, or bank transactions. The result quality of transactional queries is hard to evaluate. Often external factors (i.e. those outside the scope of search engines) are important for the users (e.g. prices, speed of service, quality).

In addition, the user may want *combinations* of the above mentioned goals. E.g., users look for a good hub (which means a starting point for further searches) or they want to know which sites exist for a certain topic.

A more detailed examination of the user goals and different methods to discover the goals of real users will be subject of the next subsection.

1.1.2 User Behavior

As seen in the previous subsection, users may have completely different goals when they search the Web. It is difficult to find out for what reasons people are typing a certain query in a search engine. Assuming the majority of users is not searching just for fun, there is a goal behind each query. Two possibilities to determine the user's goals are discussed in the following. Firstly, one can ask search engine users, e.g. on the query page to voluntarily click at a goal category (user survey). The second method is to analyze the queries logged by the search engine and try to determine the user's goal while reading the query (log analysis). Data mining approaches of log analysis can be found in Baeza-Yates (2004).

Broder (2002a) conducted both, a user survey of ALTAVISTA[2] users as well as a query log analysis. One surprising result is that a big part of users "abuse" search engines to navigate to websites they already know. Before the times of World Wide Web, a full-text search predominantly served informational goals. Table 1.2 shows

[1] www.google.com/features.html#lucky
[2] www.altavista.com

Goal	User Survey Broder 2002	Log Analysis Broder 2002	Log Analysis Rose/Levinson 2004	Log Analysis Jansen et al. 2007
Informational	~ 39 %	48 %	61.9 %	80.6 %
Navigational	24.5 %	20 %	13.5 %	10.2 %
Transactional/ Resource	~ 36 %	30 %	24.6 %	9.2 %

Table 1.2: Distribution of search engine user's goals

that following the survey almost one quarter of the users asked have navigational goals. The users express that only 39 percent of the queries possess informational character. The third goal was, like the navigation, simultaneously emerging with the Web: about 36 percent of the queries were transactionally motivated.

Users were randomly selected and the response ratio was about ten percent. A problem results from the fact that the users decided themselves whether they participate. This may somewhat bias the outcome.

The query-log analysis performed by Broder (2002a) shows results slightly different from the user survey. A noticeable shift away from the navigational tasks towards informational and transactional tasks can be observed in Table 1.2. A reason for the difference is that the original user goals are not definitely known by the analyst but the query analysis leads to assumptions of these goals. The tendency to fewer navigational queries is also supported by the log analysis of Rose and Levinson (2004). Contrary to Broder's classification performed by human beings, the latter use automated analysis methods.

Rose and Levinson have started with a manual classification of Web search queries, too. By looking at what people are searching for they tried to find out why people are searching in order to understand the underlying goals. The input was not limited to the query itself. They supplemented this information by results returned by a search engine, results clicked on by the user, and further searches or other actions of the user. Some granularity was added to the goal categories based on real-world queries analyzed. Each identified search goal is shown in Table 1.3 on the following page. The second column of the table contains a description of each search goal. A sample query is only added to the lowest goal level. An example for an "open directed informational query" (Goal 2.1.2) is "Why are metals shiny". It is not meaningful to assign examples to the higher level "directed informational query". The *transactional search* is replaced by the more general

Search Goal	Description	Examples
1 Navigational	My goal is to go to specific known website that I already have in mind. The only reason I'm searching is that it's more convenient than typing the URL, or perhaps I don't know the URL.	aloha airlines; duke university hospital; kelly blue book
2 Informational	My goal is to learn something by reading or viewing webpages.	
2.1 Directed	I want to learn something in particular about my topic.	
2.1.1 Closed	I want to get an answer to a question that has a single, unambiguous answer.	what is a supercharger; 2004 election dates
2.1.2 Open	I want to get an answer to an open-ended question, or one with unconstrained depth.	baseball death and injury; why are metals shiny
2.2 Undirected	I want to learn anything/everything about my topic. A query for topic X might be interpreted as "tell me about X."	color blindness; jfk jr
2.3 Advice	I want to get advice, ideas, suggestions, or instructions.	help quitting smoking; walking with weights
2.4 Locate	My goal is to find out whether/where some real world service or product can be obtained.	pella windows; phone card
2.5 List	My goal is to get a list of plausible suggested websites (I.e. the search result list itself), each of which might be candidates for helping me achieve some underlying, unspecified goal.	travel; amsterdam universities; florida newspapers
3 Resource	My goal is to obtain a resource (not information) available on webpages.	
3.1 Download	My goal is to download a resource that must be on my computer or other device to be useful.	kazaa lite; mame roms
3.2 Entertainment	My goal is to be entertained simply by viewing items available on the result page.	xxx porno movie free; live camera in l.a.
3.3 Interact	My goal is to interact with a resource using another program/service available on the website I find.	weather; measure converter
3.4 Obtain	My goal is to obtain a resource that does not require a computer to use. I may print it out, but I can also just look at it on the screen. I'm not obtaining it to learn some information, but because I want to use the resource itself.	free jack o lantern patterns; ellis island lesson plans; house document no. 587

Table 1.3: Search goal hierarchy (Rose and Levinson, 2004)

resource category, which comprises searches for resources other than information planned to use in the offline world.

The categorization process and its result help us to understand the space of user goals. Based on the manual categorization, Rose and Levinson defined rules for

automatically assigning queries to goals. Averaging over their three studies, they came up with almost forty percent non-informational queries (compare Table 1.2). The part of navigational queries only adds up to 13.5 %, whereas the transactional queries only come to almost one quarter of the queries.

Differences to Broder can result from the automatism as well as from the fact that the latter sampled all queries performed, whereas here only session-initial queries were regarded. Navigational sessions are likely to be shorter than other sessions.

Jansen et al. (2007) made more recent classifications of Web queries exclusively using an automatic process. They came to eighty percent of informational queries, whereas navigational and transactional queries accumulated to a little more and a little less than ten percent, respectively. White and Drucker (2007) have identified two classes of extreme users, *navigators* and *explorers*.

Knowing and understanding the user goals is an important prerequisite to fulfill them. A deeper understanding can help search engine developers to improve their products. For example, requirements for the ranking algorithm, which determines the presentation of search results, change with the user's goals. Queries with the need for advice can be answered using connectivity (link) based relevance because other users have "voted" for the best results (see Subsection 1.3.5 for ranking algorithms). For queries for an open-ended research, traditional IR measures like term frequency may create better results.

In order to fulfill the various user needs and goals, search engines access a lot of different types of information sources, which will be described in Section 2.1.

1.2 Search Tools

The information demand of an Internet user faces an enormous, more or less structured information supply in the World Wide Web. Using search tools is one way of trying to find a match between the information demand and the information supply. This section will describe different principles of search tools.

In order to quickly answer a search query, search tools make use of an already existing own or a foreign database. In this database, a part of the information available on the Web is being kept ready for quick access. Search tools can be differentiated according to the origin of data into *portals* and *result providers*. *Portals* send the search query to one (or in the special case of meta-search engines to more than one) foreign databases. *Result providers* make their search results available for the portals using own databases. In practice, there often exist combinations of both.

Search tools can also be distinguished based on the generative mechanism of their database into search engines and Web catalogs. In every-day language use, the term "search engine" is often used for both. But the way their data basis was created makes a big difference. *Web catalogs or directories* as e.g. YAHOO or the *Open Directory Project* DMOZ[3] are editorially arranged by human beings whereas *search engines*, like e.g. GOOGLE[4], produce their databases by machines. Catalogs are created by dividing webpages into one or more content related categories (which were chosen after a preselection that considers qualitative criteria). The result of this procedure is a high-quality database. The user can either look into the individual categories or carry out a full-text search through the whole database. If a category of the directory already fulfills the user's information need, he/she can obtain results without an explicit search query. Even if there are very comprehensive catalogs, these will hardly reach the extent of machine-generated data volumes created by search engines. Furthermore, the databases of catalogs cannot be kept up to date that frequently because of the high manual efforts.

Search engines create their results using a standardized automated process without any manual or human help. A *crawler* or *robot* reads, stores, and analyzes Web content. Search engines generate their data basis starting with a given set of webpages by following the hyperlinks of these pages and by thus considering certain decision criteria. The so found new pages are then added to their databases. The database is prepared by creating different indexes (e.g. a word and a phrase index) in a way that enables a quick reply of the search inquiries. The output of the search results is being presented in form of a list that is sorted according to a relevance measure in traditional search engines. Normally, this relevance is being determined by a static relevance, which is assigned to the webpage, and a dynamic relevance stating how good the result meets the search inquiry (Arasu et al., 2001b; Brin and Page, 1998; Deo and Gupta, 2001a). The static as well as the dynamic relevance – applied to the most frequently posed search queries – is being used by the VOX POPULI ALGORITHM (VPA) (Schaale et al., 2003) in order to preferably add those relevant webpages to the database for which content most users frequently search. The functioning of search engines will be described in-depth in the next chapter.

Many of the existing search tools cannot be put in only one of the above described categories. They use for example directory and machine search results in the same result page. In Figure 1.3 on the next page some of the major search tools are exemplarily assigned to one or more categories, knowing that a sharp classifi-

[3] `www.yahoo.com, www.dmoz.org`
[4] `www.google.com`

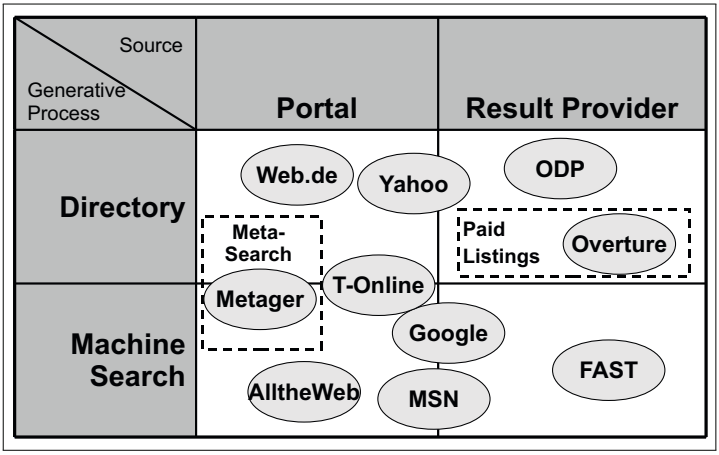


Figure 1.3: Characteristics of search tools

cation is impossible. The figure rather indicates the characteristic properties of the tools, their past development, and the importance of their business in the respective area.

Looking in the upper left field of Figure 1.3, the WEB.DE[5] and YAHOO[6] are well-known portals. That does not mean that they do not provide own results. Both started with a manually administered directory which is still actively searchable. Currently, their primary search results come from machine search. WEB.DE uses GOOGLE's machine search results whereas YAHOO uses its own search technology providing machine search results, too. Both providers are perceived mainly as information portals.

In the upper right field, we find directories with principally own results. A typical representative of this category is the OPEN DIRECTORY PROJECT DMOZ (ODP)[7]. The ODP is an open-content Web directory maintained by a community of editors. It offers a basic search interface, but it plays its main role delivering their categorization data to other search tools. E.g. GOOGLE uses ODP's results in order to add information to their machine search results. Social bookmarking tools like DELI.ICIO.US[8] may also fit in this category.

[5] www.web.de
[6] www.yahoo.com
[7] www.dmoz.org
[8] http://del.icio.us

A special case of directory results are provider of commercial advertisements. They return as results advertisements, also known as paid listings or sponsored links, which either fit to the query, or at least somebody paid for them because of any existing relationship to the query. One provider of paid listings is OVERTURE[9] (now part of YAHOO), which does not possess a search portal but exclusively delivers results to other search tools.

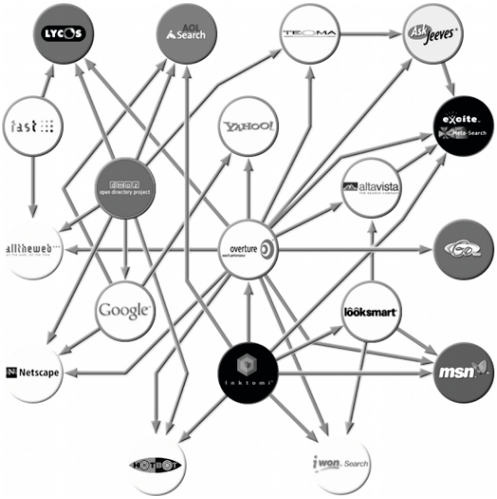
An example for a pure machine search portal without any own results is ALLTHEWEB[10]. It used to receive its results from the Web-search-technology provider FAST[11], a pure machine-search result-provider, until the latter sold their Web-search division to OVERTURE. Machine search portals with own results are MSN[12] and GOOGLE[13]. T-ONLINE[14] is a portal that returns both, machine search results (from GOOGLE) and directory results from their own information supply.

Another special case can be found in meta-search engines that send the user's query to multiple search engines or directories and aggregate the results. They consist of portals without any own results. A typical representative is METAGER[15].

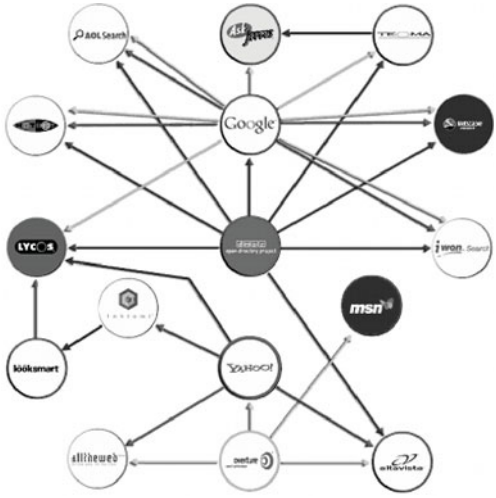
Users may not always be aware of the different types of result generation, although result quality depends on the generative mechanism and the database used. For simple needs, it may be sufficient to enter something in the search box to get appropriate results without knowing their origin. A further typology of search tools can e.g. be found in Lewandowski (2008).

The complex relations between result providers and portals are shown in Figure 1.4 and Figure 1.5. The arrows indicate the flow of results. In most cases, they point from a result provider towards a portal or another result provider. If the search tool uses external search results as its main source they are called primary results. Otherwise, they are called secondary results. The above described directory results are divided into genuine directory results and paid results.

[9] www.overture.com
[10] www.alltheweb.com
[11] www.fast.no
[12] www.msn.com and www.bing.com (belonging to MSN)
[13] www.google.com
[14] www.t-online.de
[15] www.metager.de



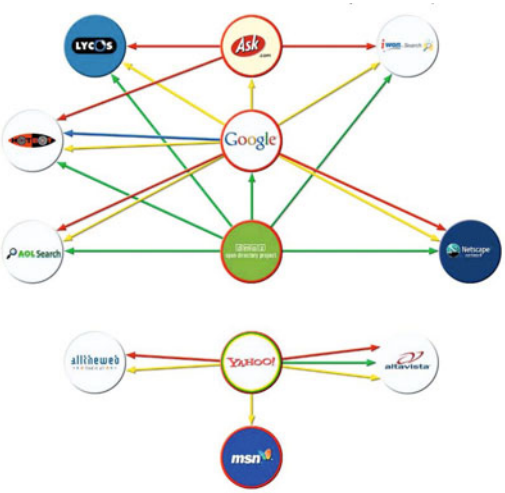
(a) 2002



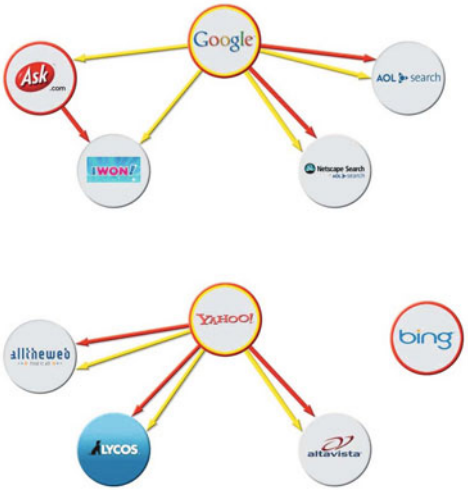
(b) 2004

LEGEND
SUPPLIES → RECEIVES PRIMARY SEARCH RESULTS
SUPPLIES → RECEIVES SECONDARY SEARCH RESULTS
SUPPLIES → RECEIVES DIRECTORY RESULTS
SUPPLIES → RECEIVES PAID RESULTS

Figure 1.4: Search engine relationship chart – development 2002–2004 (Clay, 2009)



(a) 2006



(b) 2009

LEGEND
SUPPLIES → RECEIVES PRIMARY SEARCH RESULTS
SUPPLIES → RECEIVES SECONDARY SEARCH RESULTS
SUPPLIES → RECEIVES DIRECTORY RESULTS
SUPPLIES → RECEIVES PAID RESULTS

Figure 1.5: Search engine relationship chart – development 2006–2009 (Clay, 2009)

In the year 2002, the search tool landscape was still complex. Results were passed in chains of up to four search tools. Looking at the following years 2004 and 2006, the picture has significantly changed. The consolidation on the search market reduced the number of actors. The bigger search companies tended to prefer own results and databases, so the number of interconnections decreased, too. In 2009, the search landscape's complexity diminished even more as shown in the lower graph of Figure 1.5b.

As the entry and search processes in directories are straightforward, the next section will focus on machine search, explaining the various steps from crawling until presentation of results.

1.3 How Search Engines Work

Various different search engines provide their services on the Internet. Although their overall goal is to help the customers with finding information on the Web each search engine uses its technology to differentiate their service from their competitors. The expression "search engine" may suggest to some users that the whole Web will be searched for their specific query directly after they have entered it in the Web browser. Such a user may cast doubts on this as soon as he is asked if he believes that it is possible for that engine to browse through the whole World Wide Web, search for his keywords and present him the ranked results after a split second. He may also wonder if somebody is able to store the whole Web content or at least huge parts of it in a searchable index. Anyway, the core competency of most search engines is not only the search and retrieve functionality, but also a management of very large databases. Search engines generate large databases containing Web information to be able to quickly answer the requests. Some providers also use external databases in addition to or instead of maintaining an own database. Existing search engines use a similar fundamental approach to generate and provide their databases and results. This leads into a general architecture of key components which is shown in Figure 1.6 on the following page.

A search engine can be divided into five modules shown in Figure 1.6: a crawler, an indexer, an analyzer, a searcher and a presenter. The crawler module typically establishes the first and only contact of the search engine with the World Wide Web data. He gathers information from the Web which will then be further processed by the indexer and analyzer module. Their results are made accessible to the searcher module in one or more central databases. The searcher answers a user's search queries using the database and gives the results to the presenter module to format a result page out of them.

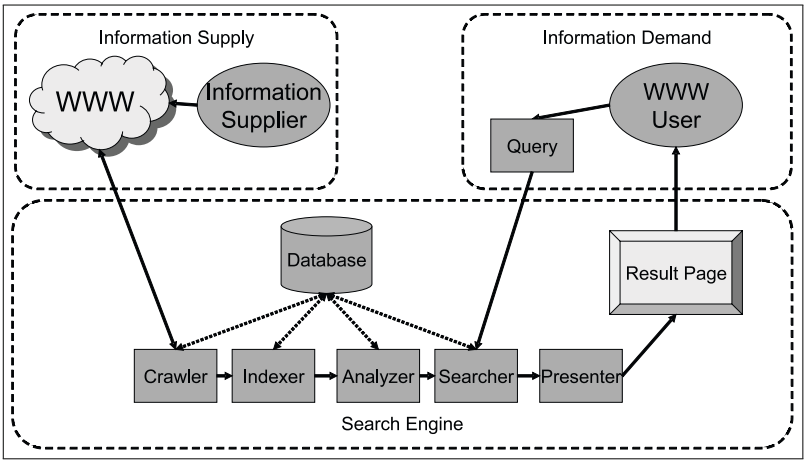


Figure 1.6: Search engine result flow

Each module belongs to one of the following two asynchronous independent processes: a back-office process which works user and query independent in the background, and a front-office process which answers the user's query at run-time. The back-office process consists of the crawler, the indexer, and the analyzer. The front-office process is performed by the searcher and the presenter. Some of the processes are allowed to access the central database. The database consists of a document repository, an inverted index, and a link database. Depending on the search engine, some additional utility indexes can be stored, e.g. a database of anchor texts.

This illustration does not necessarily mean that every search engine follows exactly the described scheme. It rather intends to describe the general functionality and the modules that are necessary to run a search engine. The following subsections will describe the single modules in detail.

1.3.1 Crawler

The first module processing the search engine result flow is the *crawler*. The behavior of the crawler in the Web is comparable to the one of a standard Web browser used by a human being. The crawler collects either as many pages as possible from the Web or the necessary amount of pages to reach his goal. Some goals of a crawler will be described in Section 2.3.

Algorithm 1.1 Simple Crawler (Baldi et al., 2003)

SIMPLE-CRAWLER(S_0, D, E)

```
Q ← S_0
while Q ≠ ∅
do u ← DEQUEUE(Q)
   d(u) ← FETCH(u)
   STORE(D, d(u), u)
   L ← PARSE(d(u))
   for each v in L
   do STORE(E, (u, v))
      if ¬(v ∈ D ∨ v ∈ Q)
         then ENQUEUE(Q, v)
```

The crawling process starts with an initial set of URLs S_0, it downloads and stores the corresponding webpages (or parts of them) and follows the containing hyperlinks according to an algorithm which may differ from crawler to crawler. Different heuristics define which sites how often will be visited. Newly-found URLs are put on a list of URLs to visit.

Crawling Process

The crawling process will be illustrated with Algorithm 1.1 which basically performs a graph visit (Cormen et al., 2001). Starting with S_0 as seed or initial set of URLs in the queue Q, the algorithm visits the webpages $u \in Q$ according to a predefined order. If one uses a first-in first-out (FIFO) approach for accessing Q (DEQUEUE), the crawler performs a breadth-first search.

The FETCH process downloads the document $d(u)$ of URL u coming from the queue. The document's content will then be stored together with its URL in the document repository D. The PARSE function collects the references to other webpages found in $d(u)$. The set of these references of URLs is called L. The link or edge (u, v) between the URLs u and v are stored in the link database E using a STORE function for all documents v in L. Those documents v that are neither in the document set D nor in the queue Q are added to the queue using the ENQUEUE function.

Crawler Problems

As the list of URLs to visit grows very quickly while traversing the Web it must be very selectively created. The search engine has to make decisions on the choice

of sites during the crawl process, because resource capacity in terms of computing power and storage is restricted. Arasu et al. (2001a) describe four key questions the crawler has to consider given the enormous size and the change rate of the Web:

1. *What pages should the crawler download?* Even though some search engines seem to aim to download the pages of the entire Web, they are only able to crawl a relatively small part of it. Some search engines specialize on specific topics or content types like news or video files. A prioritization of the pages to visit may support decisions such as which page to visit first.

2. *How often should the crawler visit the pages?* After one crawl process is completed, the crawler needs to keep its repository up-to-date. He decides upon the revisit interval of each webpage. A news site may for example change much more often than a certain content of a private picture archive. For this reason, the crawler measures past change frequencies and uses them for future activities. Using this information, it can minimize the work load by visiting seldom changing pages less often than frequently changed pages.

3. *How to minimize the load on visited websites?* As the crawler acts more or less like a user browsing along websites, it uses different resources of the website provider: On the one hand, the content provider's server performs some work answering the HTTP request or even a runtime database request, and on the other hand the bandwidth of the provider's Internet connection is used. Assuming the provider has to pay for both and several search engines access his sites a few times per day downloading either the whole or big parts of the content each time, this could cause an enormous impact on his profit and loss statement. So the provider may exclude a resource-consuming search engine from accessing his server. Consequently, the crawler needs to be run by a sensitive concept of resource consumption.

4. *How to parallelize the crawling process?* The tremendous size of the Web causes the need of distributing the crawling workload on different machines. This parallelization is necessary during the download of webpages and for further processing of the fetched data. The coordination of crawling tasks is a critical point in order to avoid double workload on the machines and – what may be even more important – to avoid visiting the same website multiple times.

Commercial search engines use several optimization techniques in order to obtain the best possible performance out of a given hardware. Computations are some-

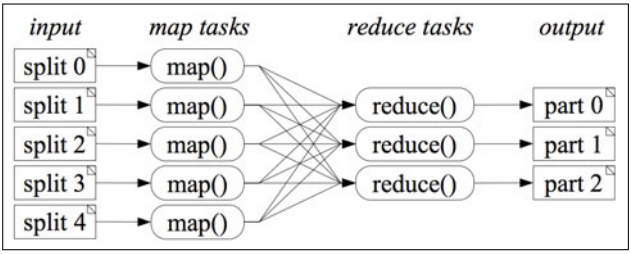


Figure 1.7: MapReduce task processing (Cutting, 2005)

times distributed over many computers in order to accomplish the task in a reasonable amount of time. GOOGLE for example uses more than 100'000 servers to run their search engine (Arnold, 2005). They use low-cost hardware (commodity PCs), working around the bottlenecks of standard operation systems by adapting Linux system to their own needs. The PCs are organized into large clusters of thousands of machines, which typically use dual INTEL processors with 4 GB of memory per machine (Dean and Ghemawat, 2008). They are connected with commodity networking hardware and store the data on inexpensive IDE disks. For these reasons machine failures indeed are common, but they are compensated by intelligent methods to handle breakdowns. A distributed file system is used to access data on the different machines (Ghemawat et al., 2003). A job scheduling system assigns tasks to the machines.

MapReduce

Dean and Ghemawat (2004) describe a programming model and an associated implementation for processing and generating large data sets on large clusters called MapReduce. The MapReduce framework addresses several issues occurring during the programming of search-engine software.

The large data amounts processed by search engines makes it necessary to run processes in parallel on multiple computers and to store data on several storage devices. MapReduce addresses the issues of how to parallelize the computation, distribute the data, and handle failures avoiding large amounts of complex code. Inspired by the *map* and *reduce* functionality of LISP and other functional languages, MapReduce prevents the developer from the need to explicitly consider these issues by himself.

It offers standardized methods to divide the computations on large data sets in a way that tasks can be processed at the same time on different machines (see Figure 1.7). The challenge is to find an appropriate partition. Running different tasks in

Offset	Log file 1	Log file 2
k_1	v_1	v_1
0	C	A
2	B	C
4	C	
6	B	

Figure 1.8: MapReduce sample input for log file analysis

parallel must not alter the solution. The output data can be wrong in the case of neglecting crucial dependencies between the tasks.

In order to use a standard framework that can be applied to different problems, the input and output data have to exist in a standard structure. The data must be organized in a data base format defined by a *key k* for each data record (corresponding to a row in a table) and a *value v* holding the data of the data set (compare example in Figure 1.8).

If the computation applied on a pair of key and value (k,v) fits in the following general scheme it can easily be parallelized and its data can be accordingly partitioned. In the first step a map operation transforms the input data into an intermediate representation $\text{map}(k_1,v_1) \rightarrow \text{list}(k_2,v_2)$. In a second step a reduce function recombines the intermediate representation into a final output $\text{reduce}\,(k_2,\text{list}(v_2)) \rightarrow \text{list}(v_2)$.

The functionality of MapReduce is explained using the following simple example. The task is to determine the access frequency of a URL during the analysis of the log files of a website. The host computer delivering the pages of a website usually stores the URL of each single request in a log file. It generates log files for certain time periods, e.g. on a daily basis. In other words, the task consists of counting the occurrences of each URL in multiple large files. Each log file contains a list of the website's URLs $u \in \{\text{A,B,C}\}$ accessed during a certain period (e.g. one day). Two sample log files are shown in Figure 1.8. In this example, the key k_1 represents the offset (or line number) in the log file and the value v_1 the line content (here the URL) at this offset.

The goal is to calculate the number of occurrences of the pages *A* and *C* represented by their URLs in the log files. Because log files can consist of many and large files, it can be necessary to process them in a parallelized way. Algorithm 1.2 shows the map and the reduce function adapted for this task.

Algorithm 1.2 MapReduce example: log file analysis

```
map(String key, String value):
   // key: file offset f
   // value: line contents u
   for each file offset f:
      EmitIntermediate(u, "1");

reduce(String key, Iterator values):
   // key: a URL
   // values: a list of counts
   int result = 0;
   for each v in values:
      result += ParseInt(v);
   Emit(AsString(result));
```

Map Step
$(k_1, v_1) \rightarrow \text{list}(k_2, v_2)$
$(0, \text{C}) \rightarrow [\langle \text{C},1 \rangle]$
$(2, \text{B}) \rightarrow []$
$(4, \text{C}) \rightarrow [\langle \text{C},1 \rangle]$
$(6, \text{B}) \rightarrow []$
$(0, \text{A}) \rightarrow [\langle \text{A},1 \rangle]$
$(2, \text{C}) \rightarrow [\langle \text{C},1 \rangle]$

Reduce Step
$(k_2, \mathit{list}(v_2)) \rightarrow \text{list}(v_2)$
$\langle \text{A}, [1] \rangle \rightarrow \langle \text{A}, 1 \rangle$
$\langle \text{C}, [1,1,1] \rangle \rightarrow \langle \text{C}, 3 \rangle$

Figure 1.9: Map and reduce step for log file analysis example

The first step is performed by the *map* function processing the log files. It analyzes the log files and produces a tuple $\langle k_2, v_2 \rangle = \langle \text{u}, 1 \rangle$ for each URL it finds in the log files. These tuples form the list(k_2, v_2). The Table "Map Step" in Figure 1.9 shows the intermediate output. This map function can now be processed in parallel on different machines. The input file or the input files are split if necessary into an appropriate number of files, which are then distributed on the machines (see Figure 1.7).

The second step is done by the *reduce* function. It adds up all values for the same URL $\langle k_2, v_2 \rangle = \langle$URL,total count$\rangle$. The output (represented by reduce $(k_2, \text{list}(v_2))$ $\rightarrow$ list(v_2)) is shown in the Table "Reduce Step" in Figure 1.9. This step can now

URL	Count
k_2	v_2
A	1
C	3

Figure 1.10: MapReduce output for log file analysis example

also be performed in a distributed way without changing the final output. The result is shown in Figure 1.10.

The principle solution steps of MapReduce (here explained by means of the simple example) can be deployed on much more complex problems, as long as they fit in the above mentioned scheme. In the search-engine context, the MapReduce programming model can be applied on the following examples (Dean and Ghemawat, 2004):

1. *The creation of a reverse link graph.* It is easy and straight forward to get to know the outgoing links of a webpage by analyzing the HTML-Code of this page. More difficult is to get to know all links pointing to a certain page, because one has to consider all pages possibly linking to that page. MapReduce addresses this problem by creating ⟨target,source⟩ pairs for each link to a target URL found in a page named source in the map function. The reduce function merges the pairs of all source URLs for each target URL to the output $\langle \text{target}, list(\text{source}) \rangle$.

2. *The calculation of a term vector per host.* For topic-specific crawling purposes the knowledge of the term vectors located on a certain host is helpful. A term vector consists of information about the webpage's content represented by its most important terms, and their frequency in the document $\langle \text{term}, \text{frequency} \rangle$. In the first step, the map function produces a list$\langle \text{hostname}, \text{term vector} \rangle$ for each input document (where the hostname is extracted from the document's URL). The reduce functions then adds all per-document term vectors for a given host producing the final $\langle \text{hostname}, \text{term vector} \rangle$.

3. *The creation of an inverted index.* The inverted index, used for basic retrieval functions, lists for a specific word or term those documents which contain this word or term. The inverted index will be described in more detail in Subsection 1.3.2. The map function analyzes each document and outputs a list of

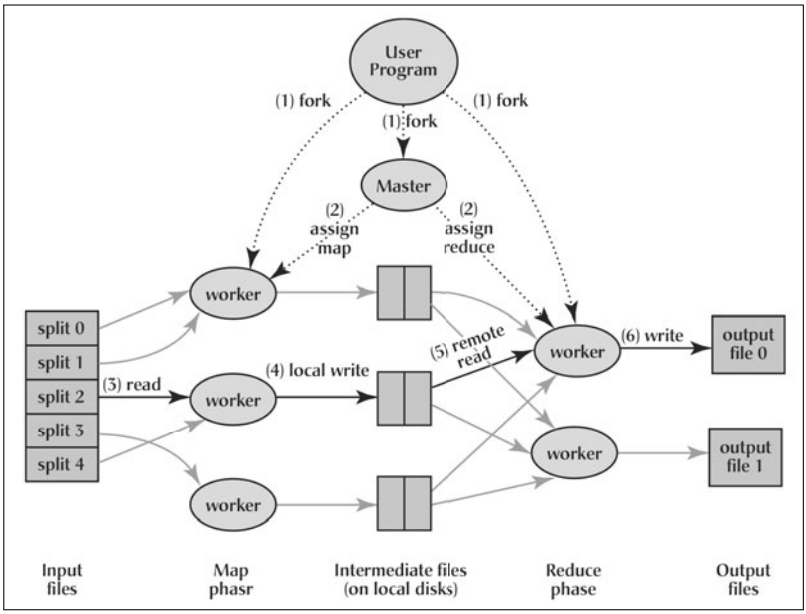


Figure 1.11: MapReduce execution overview (Dean and Ghemawat, 2008)

⟨word,document ID⟩ pairs. The reduce function sorts the documents IDs for a specific word and outputs the inverted index as ⟨word, *list*(document ID)⟩.

A search engine typically performs the tasks of the examples 1–3. The MapReduce concept is used by e.g. the search engine GOOGLE and the open-source Web-search software NUTCH[16] (White, 2006) in order to improve their operations. After adapting the tasks to the MapReduce scheme, the search-engine software can call the map and reduce functions as subroutines. MapReduce performs several steps in order to automatically split the input files, distribute the processes on different machines (workers) and concatenate the output files.

Figure 1.11 shows the steps (1–6) performed when a program of the GOOGLE system calls the MapReduce function. First, the input files (for Example 1: Web documents) are split into several (here: $M = 5$) pieces, and the *user program* (in this case the GOOGLE indexer) is started multiple times on a cluster of machines (1). One copy of the program is a master which assigns M map tasks and R reduce tasks to the other programs called workers depending on their work load (2).

[16] `www.nutch.org`

Workers with map tasks read the corresponding input data split, parse key/value pairs and process the user-defined map function (3). This output is periodically written to local disks, partitioned into *R* regions (4). The master will be notified about the data and assign a reduce worker who starts with grouping together same intermediate keys (5) and process them in the same reduce task (6). The master returns the process back to the user code.

The NUTCH development project implemented MapReduce functionality in order to improve the software's scalability (Cutting, 2005). NUTCH used to be limited to approximate 100 million pages processed on a handful of machines. Together with the implementation of the distributed file system HADOOP DFS (Borthakur, 2007), a distributed computing platform was needed to partition stages (jobs) into work units (tasks), and to allocate, start, monitor, kill or re-start these tasks if necessary. MapReduce has made the code of NUTCH substantially smaller and simpler and permits multi-billion page collections. Some extensions have been implemented for a better performance: the output can be split to multiple files in order to save subsequent input/output operations and mixed input value types are now allowed in order to save value conversions. The scalability of NUTCH was examined in more detail by Moreira et al. (2007) and Michael et al. (2007).

Ranger et al. (2007) have evaluated the suitability of MapReduce as a programming environment for shared-memory systems. They found that the simplicity of the parallel code exhibits a promising model for scalable performance also on multi-core and multi-processor systems.

Scheduling a Crawl for First Visits

The above described methods help to distribute the crawling task on multiple computers. Taking into account the enormous size of the Web, the resulting performance improvements can still not be sufficient to crawl the entire Web in a reasonable amount of time. Even if the computational task is split and processed on multiple machines, the sub-tasks still remain huge. Consequently, the crawling task itself became subject of different improvement measures. Differences in server responsiveness can be considered as well as different user needs and change ratios in order to improve the crawling process.

Several intelligent strategies have been developed to fulfill as much as possible of the crawler's goals in a given time frame. Castillo et al. (2004) distinguish between approaches for scheduling first-time visits and revisits of webpages. They have proposed and evaluated several strategies for webpage ordering during the crawling of previously un-visited webpages. The crawler has to work in the framework of the following parameters which will be the basis for restrictions:

1. The *time interval* w (in seconds) between connections to a single website should have a minimum size. Web servers usually have a limited capacity of answering requests coming either from user-driven browsers or automatic crawlers. Thus, they consider those crawlers as impolite that send too many requests per time unit. Crawlers have to control the number of requests to a single website, because the website operator may ban them from the servers in case their access frequency becomes too high. Koster, one of the first authors to point out the problems caused by Web crawlers, proposed in 1995 an interval of $w = 60$ seconds (Koster, 1995). Today, an interval of greater than $w = 15$ seconds seems to be acceptable.

2. The *number of pages* k a crawler is able to download per connection with the same website can be increased. Common crawlers use each HTTP-connection to download only one webpage at a time. The keep-alive header of HTTP/1.1 can be used to re-use an open connection and hence decrease the servers' workload. Thus, some of the time consuming opening processes can be avoided.

3. The *number of simultaneous requests* r to different websites determines the degree of parallelization of a crawler. A serial process is the result of $r = 1$. If the crawler is restricted to one connection to a given website, the number of websites to be visited in the queue is the maximum useful number of simultaneous requests r. Towards the end of a crawl, a decreasing queue can limit the crawler's performance.

4. The *bandwidth* B (measured in bytes per seconds) of a crawler is usually larger than the maximal bandwidth B_i^{MAX} Web servers can deliver their pages i with.

In the following simple case, the download time for visiting webpages can be determined straight forward. A crawler must download the pages $i \in \{1,\ldots,5\}$ with file sizes P_i. A trivial solution of this problem disregarding other limits than the bandwidth B is to download all pages simultaneously at a speed B_i proportional to page sizes P_i:

$$B_i = \frac{P_i}{T^*}$$

with an optimal download time T^*:

$$T^* = \frac{\sum_i P_i}{B}.$$

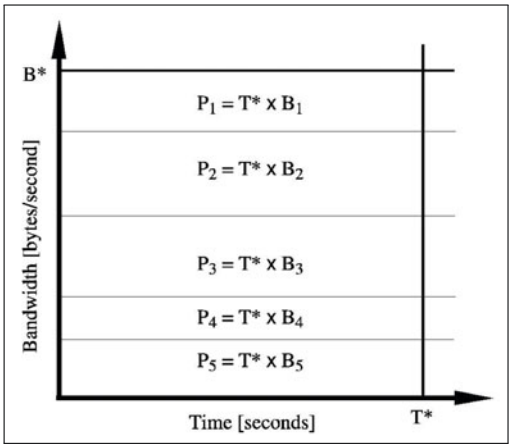


Figure 1.12: Trivial schedule of Web crawling (without restrictions) (Castillo et al., 2004)

Figure 1.12 shows a solution for this trivial case. The optimal download time, calculated as T^*, is hard to reach under more realistic constraints. Taking into account the intervals w between two downloads and assuming different maximum bandwidth B_i^{MAX} of the webpages i, gaps between the downloads may become necessary. The bandwidth restrictions and the gaps can lead to a schedule where the crawler's bandwidth is not always fully used.

The time span T_i the Web server needs to deliver page i is determined by the file size P_i and the maximum bandwidth B_i^{MAX} as

$$T_i = \frac{P_i}{B_i^{MAX}}.$$

In Figure 1.13, an example schedule is shown assuming that webpages 1 and 2 as well as 4 and 5 belong to the same website (and consequently are located on the same host server), so that the minimum time interval w has to be taken into consideration. Let us assume that the number of connections per server is limited to two. The time interval w causes a greater minimum time span T^{**}, in this case resulting from the critical path determined by the delivering times of pages 4 and

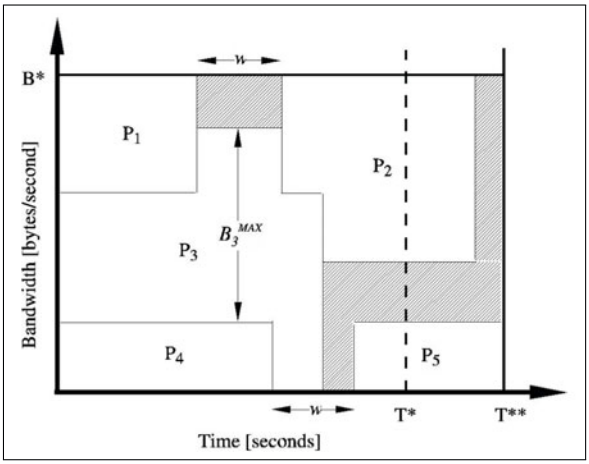


Figure 1.13: Sample schedule of Web crawling (shaded areas indicate wasted bandwidth) (Castillo et al., 2004)

5 and the minimum time interval for the common server:

$$\begin{aligned} T^{**} &= T_4 + w + T_5 \\ &= \frac{P_4}{B_4^{MAX}} + w + \frac{P_5}{B_5^{MAX}}. \end{aligned}$$

Bandwidth B of the crawler is wasted because of the maximum bandwidths B_i^{MAX} of the Web servers that deliver the pages i, in the figure shown for B_3^{MAX} during the download of P_3 (shaded areas in Figure 1.13). The bandwidth of the crawler cannot be exhausted, because the download of P_3 cannot start in parallel with P_1 and P_4 due to the limit of the crawler to two connections at the same time. Thus, the download time without restrictions T^* cannot be reached, and the download needs a time of T^{**}.

If possible, these problems can be overcome by downloading pages from preferably different websites at the same time.

Crawl Strategies

Castillo et al. (2004) have applied different crawl strategies for the usage of bandwidth and varied the restrictions of the above mentioned parameters. They used the cumulative sum of PageRank of crawled pages as an objective function for evaluating the strategies. Other importance metrics are proposed by Cho et al.

(1998). The experimental setup consists of a downloaded crawl of 3.5 million pages from over 50'000 websites. A simulator was run on a Web graph using different scheduling strategies. In a hierarchical approach, websites were queued first (long-term scheduling) and then the pages of each site were put into a crawl order (short-term-scheduling).

Three long-term strategies using PageRank-values (called "Optimal", "Batch" and "Partial") were compared with two simple strategies (called "Depth" and "Length"). The "Optimal" strategy uses precalculated PageRank values (which are not available during a real crawl) and crawls sites with high PageRank first. The "Batch" strategy calculates PageRank during the crawl process on the basis of the so-far downloaded pages (batch). The "Partial" strategy supplements this strategy with a kind of "temporary" PageRank calculated for those pages that are not downloaded yet but that are already referenced by known pages. The "Depth" strategy crawls those pages first that can be reached by following the fewest hyperlinks. The "Length" strategy crawls those sites first that have the most pages in its queue.

As one can expect, the "Optimal" strategy produces a high increase of PageRank at the beginning of the crawl. After having retrieved about 75 percent of the pages, however, the "Length" strategy becomes better and dominates even the other types of strategies.

Update Strategies

Another way of using given resources in a more economical way is to adapt the update strategy of a crawl to the above mentioned needs. During the update, the crawler revisits webpages he has already downloaded within a previous crawl. Traditional crawlers fetch pages until a certain stop criterion is fulfilled. For example, this criterion can be a reached size limit of collected data or the achievement of a predefined set of (topic specific) pages to crawl. A few days or weeks later, when the collection has to be refreshed following a time schedule, the crawler builds a new collection from scratch using the same process again. Cho and Garcia-Molina (2000) refer to this type of crawler as a *periodic crawler*.

A more economic approach to refresh the collection can be followed by an *incremental crawler* which refreshes and replaces existing pages without erasing the whole collection each time. During the incremental update, the crawler replaces outdated "less important" with new and "more important" pages. For this purpose the crawler estimates how often pages change only revisit them if their probability of having changed is high. The effect of this optimization shows up in savings in network usage and improvement in the freshness of a collection.

Other Strategies

Various strategies have been introduced for improving quality and speed of the crawling procedure. Cho and Garcia-Molina (2003) show that webpage changes can be regarded as a Poisson process. Risvik and Michelsen (2002b) estimate the refresh frequency of a document based on its change intervals in the past. Arasu et al. (2001a) describe a freshness metric and a refresh strategy which aims to guarantee a data quality for a given resource usage.

Different crawling strategies with the aim of crawling the most important pages first are presented in Cho et al. (1998). A new strategy based on the users' interests which is discovered by analyzing query logs is shown in Schaale et al. (2003). This strategy is used in the VOX POPULI ALGORITHM (VPA) and will be the subject of Section 1.4.

Focusing on certain topics is another possible way of reducing the crawler load (Chakrabarti et al., 1999b; Diligenti et al., 2000). Ahlers and Boll (2008) present crawling approaches that focus on the regional location of the users.

1.3.2 Indexer

After the crawler has downloaded and stored a set of webpages, two major modules continue processing the gained data: the indexer and the analyzer. The indexer organizes the collected webpages. It stores webpage information into an index in order to enable a later fast locating of pages during the query answering process.

The information stored can be differentiated in a *text index,* a *link index,* and multiple *utility indexes.* Arasu et al. (2001a) call the indexes *text*, *structure*, and *utility indexes*. His view of the search engine architecture is shown in Figure 1.14. Brin and Page (1998) mention not only a text index but a *document index* as well as a *lexicon*. The indexer may also build a *document repository* in order to show the (at crawl time) stored webpages to the user at query time. A utility index may for example contain anchor texts. The architecture they have used for the GOOGLE search engine is shown in Figure 1.15. Anyway, most engines maintain a document database where whole documents are stored and a separate text index for accessing the documents.

The text index contains the information necessary to answer a text-based query. For this reason the indexer extracts words and useful terms from a webpage and memorizes the URL or a document identifier pointing to the webpage. It results in very large "lookup table" that provides all those URLs of pages where a given word or term occurs (Arasu et al., 2001a). This data structure is often referred to as *inverted index* or *inverted file*. The inverted file stores a list of (all searchable)

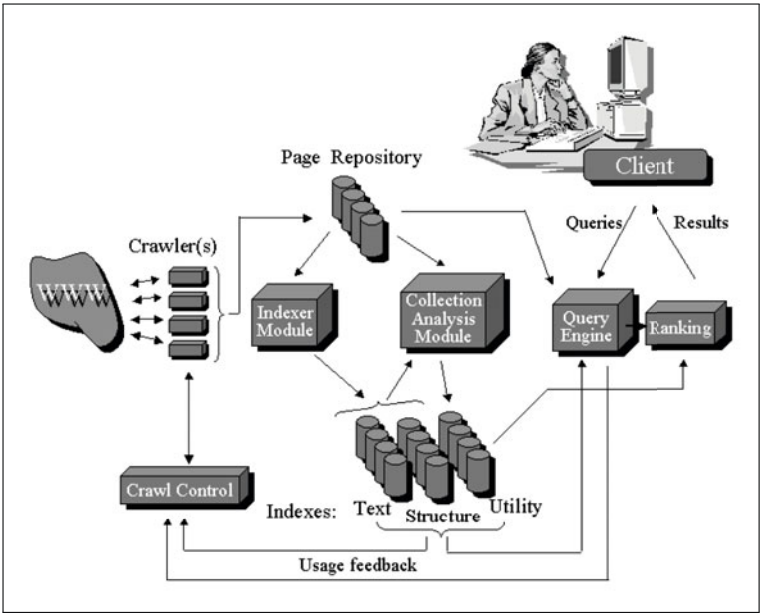


Figure 1.14: General search engine architecture (Arasu et al., 2001a)

words with pointers to the location in the data collection of each word (Risvik and Michelsen, 2002a). Brin and Page (1998) represent documents and words in the databases by document IDs and word IDs. Even though the inverted index is limited to the pages that have been crawled, the size of the Web causes large index files.

Stata et al. (2000) make use of the vector space model of information retrieval. They store vectors in a high-dimensional space of terms representing the documents. The terms are weighted by their importance in the document and in the set of documents. This representation allows the comparison of different types of documents and even of queries with documents. They have built-up a term vector database, which makes the comparison of documents and terms much faster.

The link index contains the graph representation of the Web with vertices (pages) and edges (hyperlinks). For this reason, outgoing links are identified in the fetched documents. The database stores the links between pairs of documents accessible from both directions (i.e., incoming links as well as outgoing links). The graph will be used for ranking purposes and for extracting neighborhood information.

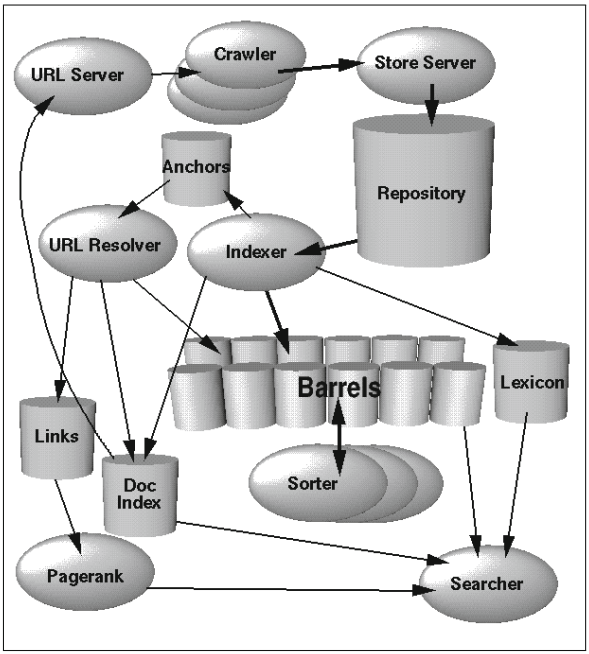


Figure 1.15: High level GOOGLE architecture (Brin and Page, 1998)

An implementation of a very large and scalable link index called the *connectivity server* is described by Bharat et al. (1998). For each URL a list of incoming links as well as a list of outgoing links is maintained.

The number and type of the utility indexes depend on the features of the query engine. There are e.g. related-page finders or a site index that stores the domain name with pages belonging to this domain. Some features like PageRank calculations are performed in a later phase by the analyzer module. The indexer also builds an URL repository to resolve the URL of a document.

1.3.3 Analyzer

Based on the databases created by the indexer, the *analyzer* performs several preliminary calculations for different purposes concerning the search, depending on the purpose and functionality of the search engine. One example is the creation and maintenance of a site index. This can be an index containing only pages of one

website or domain allowing searches on this specific site or domain (Arasu et al., 2001a).

A common application is the calculation of PageRank. Brin and Page (1998) have introduced PageRank in order to calculate a quality ranking for each webpage. The link structure of the Web is used as basis to evaluate webpages. The idea behind is to apply methods from academic literature citation to the Web. Counting citations or backlinks of a page gives an approximation of a page's importance or quality. Like in academic citation, a link or citation is seen as a vote for the page or academic paper, respectively. The number of inlinks can be used as a simple measure of quality.

PageRank extends the simple counting by a weight of the citing page, like in academic literature, where it is more valuable to be cited by a paper in an excellent journal. Furthermore, the number of links on a page will be normalized.

The PageRank measure can be used as one criterion for the output sorting of Web-search results. The calculation is recursive and will be described in more detail in Subsection 2.4.1 where different ranking methods are presented.

1.3.4 Searcher

The *searcher* (also known as *query engine* or *query processor*) receives search queries from the Web user. It scans the indexed database for documents that match the query best. The searcher looks-up the terms of the query in the inverted index and retrieves a set of document IDs that contain the queried words. Some search engines state to have more than four billions of webpages in their database. Therefore, and because of the fact that most queries consist of only one or two words, the number of returned pages can be quite huge. Large result sets follow especially from short and general queries. The presenter will later bring these documents in an appropriate order.

One crucial attribute of a good query engine is to return results within a very short time. Typical search engine users tend to follow a trial-and-error approach when formulating the query. They do not spend much energy on formulating a precise query but look what they get after their first trial. If they are not satisfied, they start the query refinement loop as shown in the classical search model in Figure 1.2 on page 8. This approach can only work if the response time of the searcher is small, i.e. significantly smaller than one second.

Rank on Result Page	Percentage of Users Viewing the Result
Rank 1	100 %
Rank 2	100 %
Rank 3	100 %
Rank 4	85 %
Rank 5	60 %
Rank 6	50 %
Rank 8	30 %
Rank 9	30 %
Rank 10	20 %

Table 1.4: Organic search results viewed on result page by rank (Sherman, 2005)

1.3.5 Presenter

The *presenter* is responsible for the way of presenting the results. Search engines present their output on so-called result pages (search-engine result-page, SERP). In the early days of Web search a query's result set was quite manageable because of the small amount of existing websites. Solely text based search engines sorted the results using the frequency of the query term appearing in the documents. A list of webpages was the result of the query. Higher weights were applied to keywords (or query terms) appearing in the headers of webpages. Typical users were already pleased if they found anything at all concerning the specified topic. With an increasing number of webpages in the Internet, the quantity of results has augmented. Thus, the presentation of Web-search results has become the new challenge for search engines.

The presenter may get a huge number of results back from the searcher. Because the searcher does not "know" about the user's information need, he takes all results into account that fit to the search query. Thus, the probability for the expected results to be among the delivered results will be higher. However, the probability for the user to actually find the expected results will decrease with a higher quantity of results. Even experienced search engine users who are able to use intelligent search queries are sometimes frustrated with the quality or order of the presented results. In order to prevent the emergence of an information overload for the user, the presenter reduces the set of answers obtained. Different strategies are used for this reduction process. Various search engines deliver very different kinds of output even if the same query has been used.

Traditional search engines yield their results as a sorted list, which – in the best case – presents its most suitable results on the top of the list. In the worst case, the user has to flip through many pages in order to find the results he/she is looking for. The number of results per page is configurable in most search engines. Anyway, the first result page is the most important because many Web users do not look beyond this page. For them, it matters most which pages are listed on top of the list. It is less important whether ten thousands or one million results are available. Eye-tracking studies on search result pages show that only the first three ranks are viewed by all users (Sherman, 2005). Table 1.4 shows that the ranks from six downwards are only viewed by at most fifty percent of the users. On a GOOGLE result page, most users looked at results in an "F" shaped scan pattern. Their eyes traveled vertically along the left side of the result page looking for relevant words and then scanning to right. The resulting pattern is also called a "golden triangle" of search and shown in Figure 1.16 on the facing page.

Thus, the sorting of the results (ranking) is playing an essential role during the presentation of results. In the case a query leads to results belonging to different topics, it may happen that the results related to one of these topics are considered to be more relevant than the other topics' results. If this topic has many relevant results (e.g. more than 100) ranked on the top, the user will hardly get to know about the results of other topics. Ranking methods are discussed in more detail in Subsection 2.4.1.

Another kind of result presentation is shown by *clustering search engines*. They divide the results into thematic groups in order to facilitate the user's navigation through a bigger amount of search results. Examples are VIVISIMO and KARTOO[17]. The functionality of clustering in the Web is the subject of Subsection 3.2.

1.4 A New Approach to Relevancy

In this section, we introduce an algorithm improving the relevancy of Web search results. The main idea is to extend existing crawling algorithms by a component that reflects the users' interests more than existing methods. The VOX POPULI ALGORITHM (VPA) creates a user feedback to the content of the searchable index (Schaale et al., 2003). The feedback derived from user query analysis is used to modify existing crawling algorithms. The VPA controls the distribution of crawler resources in a way that privileges content with stronger user demand.

[17] `www.vivisimo.com, www.kartoo.com`

Figure 1.16: F-shape of search (Sherman, 2005)

The basic components for the mathematical description of user interests, result relevancy and the crawling process are defined as follows. Users of search engines express their interests through queries $\vec{q}_i$, which they address to a searchable database (index). Assuming the users address H queries consisting of I keywords q_i $(1 \leq i \leq I)$ to this index, the query will be presented as vector

$$\vec{q}_h = (q_1, ..., q_I)_h \quad (1 \leq h \leq H). \tag{1.1}$$

I is the query length in words. The average number of keywords per query is $I' \approx 2$.

The objects users are searching for are documents d_j $(1 \leq j \leq J)$ (HTML pages, tables, text processing documents, ...) containing information. These documents are grouped (organized) in K domains D_k $(1 \leq k \leq K)$ presenting sets of documents

under a common editorial responsibility and address:

$$D_k = \bigcup_{j \in J_k} d_j \quad (1 \leq k \leq K), \tag{1.2}$$

where $J_k \subseteq \{1,\ldots,J\}$ holds the numbers of those documents belonging to a common domain D_k. The number of domains is about $K' \approx 12$ millions in Germany (DENIC, 2008) and the number of documents per domain is in the range $1 \leq |D_k| \leq 10^8$. The number of domains K is growing by approximately one million per annum.

Each document d_j contains searchable information. In general, the searchable information is limited to text information. Content that is hidden for current search technology in non-indexable formats (bitmaps, scripts, etc.) will be neglected here and in the following (for hidden Web or deep Web see Section 2.2). For the evaluation of the content it is not only important *which* keywords occur in a document but *where* in the document they occur. Keywords can be located in the body text of a document as well as in headers, in metatags, in tables, in links or in other HTML format elements of the document. Thus, a document d_j is characterized by the content of keywords q_i $(i = 1,\ldots,I)$ and their position in certain format elements e_l $(l = 1,\ldots,L)$.

During the crawling and indexing process, the image $\hat{d}_j$ of a document d_j in the searchable index (with $\hat{J}$ documents in the index)

$$\hat{D} = \bigcup_{j=1}^{\hat{J}} \hat{d}_j \tag{1.3}$$

contains a reduced set of information – the keywords and their position in the format elements of the document. When a query is addressed to the index $\hat{D}$, the ranking algorithm generates a set of documents (links) which is ordered by the relevancy of the documents found. In order to describe the document ranking process which generates a result set for each query, the density $\rho_{i,l}^{j}$ of keywords q_i within the format elements e_l of a document d_j is introduced:

$$\rho_{i,l}^{j} = \frac{n_{(q_i,e_l)}}{n_{e_l}}, \tag{1.4}$$

where $n_{(q_i,e_l)}$ is the number of the occurrences of the keyword q_i in the format element e_l and n_{e_l} is the total number of words in this format element. There are two basic ranking methods – the dynamic and the static ranking. Their distinction

is named according to the ranking's calculation point in time. The dynamic rank is (dynamically) calculated at search time whereas the static ranking is already calculated at crawl time (and remains static from this time).

The dynamic rank of a document depends on two factors only – the keywords q_i of the query and the content of the documents. This means expressed in a "thumb rule": the higher the keyword density in the document the higher is the dynamic rank of this document. The relevancy function $R^d(q_i)$, defining the dynamic rank of a document for a query q_i, can be written as:

$$R^d(q_i, d_j) \sim \sum_{l=1}^{L} \mu_l \, \rho_{i,j}^l \tag{1.5}$$

for a single keyword query. The coefficients μ_l are free parameters, defining the importance or weight of each format element e_l. For example, the occurrence of a keyword in an URL is usually much more important than in the document text itself ($\mu_{URL} > \mu_{text}$). Usually, these functions become modified for different purposes such as suppression of unwanted information (spam).

The practical work on search engines has shown that using only a document related, dynamic ranking is insufficient. In order to also include the importance or the popularity of a domain (popularity among the webmasters, not necessarily among Internet users), a new type of algorithms was invented – the static ranking (Brin and Page, 1998). The static rank R^s of a document $d_j \in D_k$ is related to the importance of the corresponding domain D_k the document is located underneath. The idea of the static rank of a domain D_k can be expressed in the following form:

$$R^s(D_k) \sim \sum_{j \in inlinks(D_k)} R^s(D_j), \tag{1.6}$$

where the $R^s(D_j)$ is the static rank of the sites linking to the domain D_k. A detailed description of static ranking can be found in Subsection 2.4.1.

The resulting rank of a document is a function of the dynamic rank (1.5) and the static rank (1.6). There is no unique or even optimal way of constructing this function. A reasonable way is to choose the resulting relevancy R^{ds} as a product of the dynamic and static rank:

$$R^{ds}(q_i, d_j) = R^d(q_i, d_j) \cdot R^s(d_j). \tag{1.7}$$

Using the product operator, a higher dynamic rank as well as a higher static rank will increase the resulting relevancy. Analyzing (1.7) a usual approach would be

using $R^s(D_k)$ instead of $R^s(d_j)$. In practice, the static component of a document does not only depend on the static rank of the domain D_k containing $d_j \in D_k$, but also on the document's position in the domain (link topology of the domain). At present, these kind of ranking algorithms are in use in every major commercial Web search engine.

The algorithms described above do indeed meet the needs of the users. This approach is reasonable from an academic point of view and it has produced remarkable results in the past. Today it has become more difficult to make use of the link topology – very often the links are not set according to the content relevancy, but for other (i.e. economic) reasons. To the extent that the search engines have become the most important information retrieval tool, they have also become a target of spamming (site owners try to fake the search engines, virtually presenting more important content than there really is). Applying filter mechanisms and modifying the parameters of the dynamic and the static relevancy calculation, one can "fine tune" the quality of the Web search engines.

The two methods described above explicitly do not take into account the most important factor, the interest of the users searching for information. The dynamic and the static relevancy of a document are influenced by the content of the site and by the "citation" by other sites. There is no methodical component that reflects the voice of the searching people. This will be done by the VOX POPULI ALGORITHM (VPA, people's voice) described in the following.

The main idea of the VPA is to use the information that is extractable from the user query analysis to enhance the quality of the search. This can be done in two different ways, by modifying either the ranking or the crawling algorithm. The VPA focuses on the crawling algorithm. The crawling algorithm defines which domain and how much of the content will be included into the search index. Sites that are not included cannot be found by the best ranking algorithm. At present, only a small fraction ($< 10\%$) of the websites is indexed by commercial search engines. The much bigger part of the Web is not visible in any of the search engines.

The source of information is the analysis of the queries $\vec{q}$, reflecting the users' interests and needs. The query set Q may contain all single and multiple keyword queries of the users (1.1). Based on these queries a multidimensional array Ω with the dimension I_{max} can be defined, containing the frequencies of all keyword combinations:

$$dim[\Omega(Q)] = I_{max} \tag{1.8}$$

I_{max} is the maximum length of a query – theoretically it can be infinite. Practically, the amount of queries having $i > 6$ keywords is less than 1%, while the average

query has about $i = 2$ keywords. In order to simplify the further calculations one can reduce the dimensionality of (1.8) in the following way:

$$\Omega^{I_{max}}(Q) \rightarrow \tilde{\Omega}^{I=2}(Q) \equiv \Omega. \tag{1.9}$$

In this reduction algorithm, the queries with more than two keywords are replaced by two-keyword queries, containing all possible paired combinations. For example, a three keyword query is equivalent to three two-keyword queries and so on.

The matrix Ω is a frequency matrix of all keyword combinations of the query set Q that is analyzed. It contains in column i and row g the frequency of the keyword combination (q_i, q_g). Ω is a symmetric matrix.

The analysis of the keyword order shows a statistical asymmetry for the keywords' sequence, i.e. $N(q_1, q_2) \neq N(q_2, q_1)$, where N denominates the number of queries. Users interested in the explicit order of the keywords can use the option called "exact phrase", which is available on any modern search engine. Therefore, it is reasonable to assume that the order of the keywords is not important for the users in the case they make simple queries. More than 90 % of all queries are of this type. Here, the approximation $(q_i, q_g) = (q_g, q_i)$ will be used.

One can calculate the eigenvectors and eigenvalues of Ω, transforming it into its diagonal form:

$$C^{-1}\Omega C = \Omega^{diag}. \tag{1.10}$$

For the details of the diagonalization procedure, see Bronstein and Semedjajew (1981) or any other standard textbook on mathematics. It is now important to understand the practical meaning of the matrices C and Ω^{diag}.

The matrix C consists of eigenvectors which are interpreted as weighted keyword combinations:

$$C^T = \begin{pmatrix} \vec{c}^1 \\ \vec{c}^2 \\ \vec{c}^3 \\ \dots \end{pmatrix} \tag{1.11}$$

where each eigenvector has the coordinates

$$\vec{c}^h = (c_1^h, c_2^h, \dots). \tag{1.12}$$

The coefficients c_i^h are positive numbers, giving each keyword q_i (similar to the definition 1.1) some "weight" compared to the other ones ("How frequent do the users ask for this keyword?"). The coefficients determine the relative importance

of a keyword within an eigenvector. A typical eigenvector (or better "eigenquery") assigns weights to the keywords of a query, e.g. a query "*free mp3 downloads*" can consist of weights like $c_1^1 = 1.00$ for the keyword "*mp3*", $c_2^1 = 0.71$ for the keyword "*downloads*" and $c_3^1 = 0.23$ for the keyword "*free*".

This query shows how the *average* user is asking, when he is searching for mp3 downloads at no costs. For an intuitive understanding, the matrix Ω is normalized to Ω' containing the relative frequencies of each term combination summing up to 100%. The reduced (N=3) keyword matrix of the example above has the form:

$$\Omega' = \begin{pmatrix} & \textit{mp3} & \textit{download} & \textit{free} \\ \textit{mp3} & 37.2\% & 8.8\% & 2.7\% \\ \textit{download} & 8.8\% & 19.2\% & 3.6\% \\ \textit{free} & 2.7\% & 3.6\% & 13.4\% \end{pmatrix}.$$

The difference between the typical keyword search at present and this approach is that the words here have different weights, determining their relative importance for the users expressed by the relative query frequency.

Additional important information about the significance of keyword combinations is contained in the matrix Ω^{diag}:

$$\Omega^{diag} = \begin{pmatrix} \lambda_1 & 0 & 0 & ... & 0 \\ 0 & \lambda_2 & 0 & ... & 0 \\ 0 & 0 & ... & ... & ... \\ 0 & 0 & ... & \lambda_{N-1} & 0 \\ 0 & 0 & ... & 0 & \lambda_N \end{pmatrix}. \tag{1.13}$$

Each eigenvalue λ_i corresponds to an eigenvector in (1.12). The eigenvalue can be interpreted as the importance of the corresponding eigenvector – it defines the importance of a query for the users.

Finally, tools have to be defined how a search engine can use the information of the user queries to determine which content should be enhanced or reduced in the index $\hat{D}$. Based on the described algorithm it is possible to determine which content is the "most wanted" content and which sites deliver this type of content:

$$(c_1, c_2, ...) \;\rightarrow\; \text{search engine} \;\rightarrow\; \text{list of ranked domains.}$$

Crawling the Internet, each domain are given certain resources of the search engine, such as CPU time and memory in the index (alternatively also the number of crawled documents or other parameters, depending on the settings of the search engine).

Algorithm 1.3 VOX POPULI ALGORITHM (VPA)

1. Generate a ranking of domains by addressing the users' eigenqueries (1.12) to the existing searchable index and determining a crawl priority for each domain according to the eigenvalues (1.13).
2. Adapt the list of crawler resources to be spent for each domain with respect to the ranking.
3. Crawl the Web according to the modified crawler resource distribution. Create a new searchable index.
4. Repeat the cycle.

The practical realization of the VPA as an extension of an existing Web search engine can be performed using the procedure outlined in Algorithm 1.3. It assigns priority values to domains. These priority values will be used to decide which domains will be crawled firstly and which domains will be crawled at all.

In order to determine which sites best fit the eigenqueries, it is useful to calculate a dynamic rank for a whole domain, not just for a single document. A simple method is to summarize the total rank of all documents in one domain:

$$R^{D_k}(q_i) \sim \sum_{j \in J_K} R^d_j(q_i). \tag{1.14}$$

Assume that the amount of resources (CPU time, number of documents, data volume, etc.) assigned to each domain during the crawling process can be expressed in a function M, with

$$M = M(D_k, R^s, \ldots). \tag{1.15}$$

In order to apply the VPA one can modify (1.15) in the following way:

$$M \to \hat{M} = M \cdot R_{VPA}. \tag{1.16}$$

The function R_{VPA} defines the VPA-correction with regard to the old crawling algorithm. The function R_{VPA} can be presented in different ways. The basic requirement concerning the function is that it is monotone concerning the parameters λ_i which quantitatively define how relevant a query for the users is. Following Occam's principle of simplicity (Entities should not be unnecessarily multiplied.) this

function should only use a minimum set of free parameters, which will allow the adoption (or "fine tuning") the algorithm to the local requirements:

$$R_{VPA}(D_k) = \left(1 + \alpha \cdot \lambda_k^{\beta}\right) \quad \alpha, \beta > 0. \tag{1.17}$$

The parameter α and β can be freely chosen as a "tuning" parameter. In the limit, the new algorithm generates the existing results in (1.16)

$$\lim_{\lambda_i \to 0} \hat{M} = M.$$

In this section has been shown how the analysis of queries can be used to enhance the relevant and "most wanted" content in a search index. This way the subjective relevancy of the search results should grow – the users will find more of what they are searching for. The existing system of the relevancy ranking of documents or domains can remain unchanged. The algorithm's aim is not to replace existing crawling and ranking algorithms, but the VPA can make Web search more relevant and more efficient.

1.5 Evolution of Web Search Engines

Since the emergence of commercial and non-commercial Web search engines in the nineties of the last century, a tremendous evolution has taken place. This considers the quality of results, the manner the results were created and also the scale in terms of size of the Web as well as the quantity of indexed pages.

Search engines have moved away from purely text-oriented information retrieval towards trying to satisfy the need behind the query (see Subsection 1.1.1). This implies a shift from the syntactic response on a query to the point of a semantic analysis of a query, of websites, and the user's context. In the following, three different stages of search engines will be identified (Broder, 2002a) and an outlook to a possible fourth generation of search will be given (Broder, 2006b).

First Generation

The first generation of search engines analyzed mainly on-page data like text, formatting and meta-tags. Players on the search market where ALTAVISTA, EXCITE, LYCOS, and others. They have basically adapted textbook algorithms of classical information retrieval to "large" corpuses.

The retrieved pages were ranked using word-frequency measurements. This was state of the art in the years 1994 – 1997, and these algorithms performed pretty

well in answering informational queries. But the use of on-page data for ranking has opened many opportunities for the webpage creator to influence the ranking. From the point of view of search engines and search engine users, this influence can negatively bias the ranking quality. This influence on ranking is from a certain extend on perceived as spam. It was one trigger for the development of the next search engine generation.

Second Generation

The second generation of search engines added off-page, Web-specific data to their systems. They used link information, anchor-text and click-through data to get information about a webpage. All this data is not part of the regarded page. It can be seen as a kind of meta-data, i.e. data about data. The anchor-text of a link pointing to a webpage can for example be interpreted as a description of the content.

Taking e.g. "big blue" as nickname for the company IBM, you will rather find these two words in links pointing to IBM's webpage than on the webpage itself. Sophisticated mathematical methods were introduced for the connectivity analysis of webpages (see 2.4.1).

These techniques were made popular by GOOGLE in 1998 – 1999, and were quickly adopted by almost every other search engine. The second generation is capable to answer both informational and navigational queries.

Third Generation

The third generation of search engines tries to answer the "need behind the query" by using multiple data sources. Several, sometimes simple but effective methods are implemented to find out what the users may want. This generation is subject to a continuing improvement process.

For example, if you type in "New York", it understands this term as a town's name and presents maps, weather forecast or hotel offers. A different treatment is applied to queries containing personalities (triggered on names), cities, medical info, stock and currency quotes (triggered on stock or currency symbol), or company names. After language detection, different ranking methods are used for different languages. Several user helps like spell checking, query refinement, query suggestion are offered.

Besides informational and navigational queries, also transactional queries are supported. This generation has moved away from syntactic to semantic matching of the query, but it still relies on the input in form of a query.

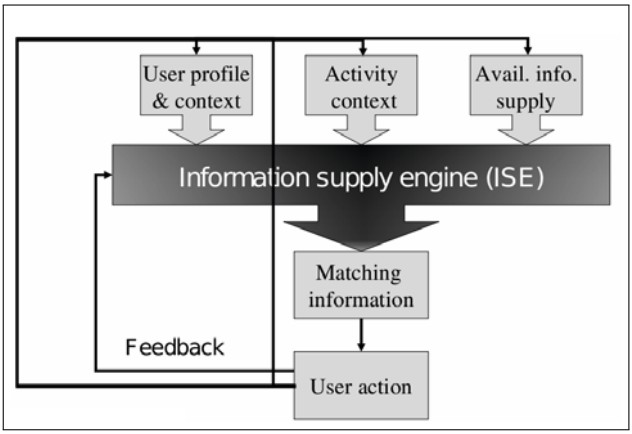


Figure 1.17: Fourth generation of search: information supply engine (Broder, 2006a)

Fourth Generation

The fourth generation does not need an explicitly formulated query. An increased usage of the user's context aims to substitute the explicit demand for information by an active information supply. As shown in Figure 1.17, three categories of input run into an information supply engine (ISE): the user profile (e.g. age, gender, ...) and user context (e.g. the location of the user), the activity context (e.g. previously performed activities of the user) and the available information supply (e.g. a searchable index). The information supply engine matches these categories in order to deliver something the user might need. The user action or a direct user feedback is used to evaluate the quality of answer. Both will be used to improve the current and future information supply.

An example is contextual advertisement, that is shown on news sites and that is related to the articles a user has chosen to read. Broder (2006a) differentiates the current approaches by the kind of needs they fulfill: *Recurrent needs* are for example fulfilled by subscriptions and news alerts. *Temporary needs* can be fulfilled by showing accessories, commentaries or related purchases on e-commerce sites, contextual help on personal computers, automatic annotations on news and others. It makes the information supply more convenient than information retrieval, because the user does not have to type a query anymore.

Anyway, it is still not easy to identify the user's need, and consequently the offered information sometimes does not match the need at all.

2 Web Structure

The relation between providing websites and searching for information in the World Wide Web can be compared to a market place in economic terms. The two sides of a market, *supply* and *demand,* can be identified. The information supply consists of the billions of companies, organizations, administrations, and individuals that offer almost any kind of information on the Web for multiple reasons. On the demand side, there are people with very different interests looking for information. Their motivations and informational needs were already described in Subsection 1.1.1.

Search Engines provide very popular means to match the user's demand with the information supply. Information Retrieval methods can help to describe and analyze the structure of the Web (see e.g. Manning et al., 2008). The Web's role as information supplier will be regarded in this chapter by first looking at the Web content, then the Web graph and finally methods of link analysis.

2.1 Evolution of Information Supply

The World Wide Web (simply referred to as the *Web*) is a large repository of all kinds of different data. The data is structured around hypertext documents consisting primarily of text as well as of links to other documents (hyperlinks).

Building of the Web started in the late eighties of the last century. Tim Berners-Lee is acknowledged to be the founder of the World Wide Web. At the CERN, the institution he worked for at that time, he wanted to simplify finding of different types of data such as reports, experimental data, electronic mail address lists and many other sets of data. The different locations were indeed already connected by the means of Internet, but until then, there did not exist a common method to access the data in the heterogeneous network environment. In order to cross the borders between different computer systems and network structures, he created and implemented a *hyperlink protocol* and a corresponding user-interface called browser (Berners-Lee and Cailliau, 1990).

Even though the concept of hyperlinking had already somewhat existed for a much longer time in the form of academic citation, the connection with the Internet enabled a direct and automatic access to the linked documents. The development

of HTML, a markup language for rendering hypertext, and of HTTP, a hypertext transport protocol, in combination with the user-friendly graphical user-interface (GUI) MOSAIC, formed the basis of the widespread adoption of the World Wide Web.

The basic concept behind hypertext is forming a web of information nodes instead of a traditional hierarchical tree or a sorted list. Section 2.2 "Web Content" will regard the different types of content on the Web. The focus of Section 2.3 "The Web as a Graph" will be put on their connecting links.

2.2 Web Content

The World Wide Web is being created by millions of individuals around the globe. Besides the variety of professionally designed and maintained websites, actually every piece of data made publicly available can become part of the Web. Anything you can store on a hard disk will be part of the Web as soon as there is a hyperlink pointing to it. In addition to the static data, dynamic streaming techniques like voice, video, or other data streams can become part of the Web even without being statically stored.

The focus of many research activities lies on the structure of the Web rather than on its content. Nevertheless, it is worth to look at the types of content. A categorization of Web content can be performed using the subgroups of *informational* and *resource* content presented in Table 1.3. But its output strongly depends on the goals of the specific user who is in the focus. This approach is limited by the fact that the same Web content can satisfy another goal category if you consider a different user. The same webpage may as well answer a *closed question* as it may contain a list of *interesting suggested websites*.

Whereas in the early days of the Web, it consisted of mainly static HTML-pages, soon a trend evolved towards documents generated at run-time. Today, only a small part of Web content consists of static HTML-pages. Already in 1997, studies conducted by Lawrence and Giles came to the result that eighty percent of the Web content has been dynamically generated, and this number is assumed to be increasing (Lawrence and Giles, 1998). The dynamism is one of the reasons why search engines have to try hard to keep up with the size of Web. The goal to build an index for almost the entire Web turns out to be a moving target, which is not necessarily achievable by the centralized architecture of existing search engines.

In order to gain an idea of the total size of the Web, Lawrence and Giles analyzed the overlap among different engines. For this purpose, they did not consider those documents that are not easily accessible. These documents are e.g. only

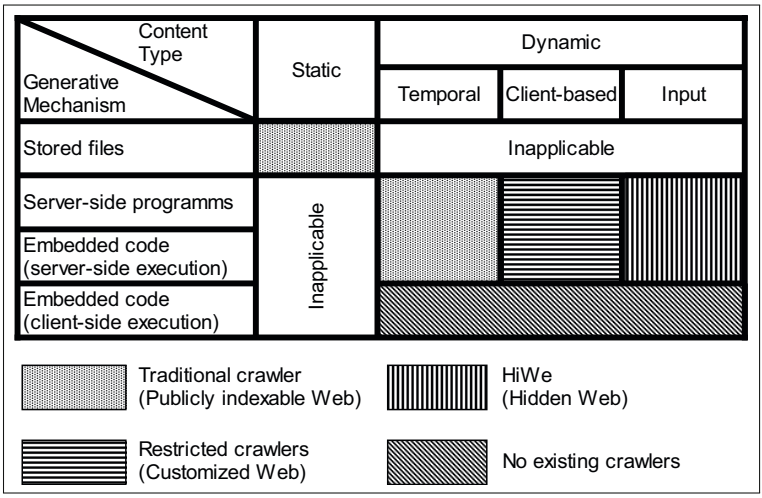


Figure 2.1: Classification of Web content based on the impact on Web crawlers (Raghavan and Garcia-Molina, 2001)

presented after filling out a search form on a webpage or they are marked as not available for search engines in a `robots.txt`-file using the *robots exclusion protocol*[1] or an authentication is required in order to access them. This part of the Web is not reachable by only following hypertext links. It is often referred to as the *hidden Web* or *deep Web* to draw a distinction to the *indexable Web* captured by search engines. In the hidden Web, a large amount of high-quality information is assumed to be provided online by organizations (like Census Bureau, Patents and Trademark Office, News media companies) using a Web front-end to access their database. If the content cannot be reached by hyperlinks, traditional search engines are not able to find it.

Two fundamental problems arise crawling the hidden Web. First, the hidden Web makes up a non-negligible amount of the whole Web. Bergman (2001) estimates that public information available through searchable online databases (that means invisible for crawlers) is 400 to 550 times larger than the static Web, which consists of seldom or never changing pages. Second, the interfaces to access these databases are intended for use by humans and, consequently, not easily accessible by robots. Thus, a big part of the Web is not covered by traditional search engines.

[1] see `www.robotstxt.org`

Within the context of crawling the hidden Web, Raghavan and Garcia-Molina (2001) have analyzed Web content by its *type of dynamism* as well as based on its *generative mechanism* used to implement the dynamism. A page is defined to be *dynamic* if a part or all of its content is generated at run-time. This means that the HTML page is generated *after* the request of e.g. a user or a search engine, whereas the HTML code of a *static* page is stored in files already *before* the request (compare Figure 2.1) independently of any user action. For example, news sites aim to present different content to the same user if he returns to the website on the same day in order to give the impression of freshness. Backstrom et al. (2009) describe approaches how to present fresh content to news site visitors.

The types of dynamism are derived from common motives for constructing Web content in a dynamic way: information that change over time *(temporal dynamism),* customization of the content for different users *(client-based dynamism),* and information that depends on a user's input *(input dynamism).* Figure 2.1 differentiates in its columns static from dynamic content and assigns crawling strategies to the types of dynamism as follows.

First, a webpage showing information that depends on the time of reading (like stock tickers or news headlines) features *temporal dynamism.* This means that requests of the same page at different points in time may lead to different results. Whereas today's search engines do index temporally dynamic content, the key question is how old the content of their databases is. Lewandowski et al. (2006) have analyzed and compared the freshness of different search engine's databases. Cho and Garcia-Molina (2003) propose some methods to estimate the frequency of change and apply the results for improving Web crawlers.

Second, webpages showing information that is custom generated for a particular user feature *client-based dynamism.* The user is identified, e.g. by cookies or his login, in order to adapt the content, design, or behavior of the website to his anticipated need. Typically, a website "remembers" settings a user has made during his last session, it may recommend products that are similar to the ones the user has already purchased, or it tries to identify a suitable language using the IP-address or the browser settings. While many of the pages generated in such way are obviously not useful for all users, it may still be helpful to crawl the same Website multiple times, i.e. in different languages. This type of webpages is addressed by a *restricted crawler* which will be equipped with cookies and/or user names and passwords.

Third, webpages returning information depending on the input of a user feature *input dynamism.* This can be the case when querying an online database. The content of the hidden Web falls in this category. The methods of Raghavan and

Garcia-Molina (2001) focus on crawling this content. The three types of dynamism described above can occur on the same webpage, e.g. an AMAZON[2] page may contain book recommendations based on the user profile (user-based dynamism), new publications (temporal dynamism) as well as the results of a user's book query (input dynamism).

Dynamic webpages can be generated by a number of different methods and mechanisms, which are divided in the following three categories (compare Figure 2.1): *Server-side programs* and *embedded code with server-side execution* both produce either complete HTML-pages or static HTML-text with dynamically generated portions of a page. Common Gateway Interface (CGI), Java servlets, Active Server Pages (ASP), JavaServer Pages (JSP) or PHP Hypertext Processor (PHP) are typical technologies to create pages at run-time. As their outputs do not technically differ from static HTML-pages, they do not exceptionally challenge traditional crawlers, once the page has been downloaded. Thus, if these pages only exhibit temporal dynamism, they belong to the *publicly indexable Web*. As soon as they are created with client-based dynamism, they are only accessible by restricted crawlers and belong to the *customized Web*. Webpages generated by server-side programs or embedded code with server-side execution whose content exhibits input-dynamism belong to the *hidden Web*. The *embedded code with client-side execution,* by contrast, may require special runtime-environments on the client-side machines. In this case, a crawler trying to emulate a Web browser has to execute the code by himself (e.g. using a Java Virtual Machine, JVM) which heavily complicates the crawling process.

Whereas for the last type (embedded code with client-side execution) a public available crawler does not yet exist, the customized Web and the hidden Web are being automatically crawled, even though for the latter a human assistance in the crawling improves its quality significantly.

2.3 The Web Graph

Already in the first design paper of the World Wide Web, Berners-Lee and Cailliau (1990) have defined – as one of its key properties – that Web elements are connected with other Web elements using hyperlinks. Thus, the Web can be modeled as a directed graph in terms of graph theory, considering hyperlinks as edges and webpages or websites as vertices. The resulting graph is only partially static: A huge part of the links and webpages do not change very often, whereas other

[2] `www.amazon.com`

pages may change their content as well as their links very frequently. That means in terms of graph theory, that not only the values assigned to vertices and edge weights may change, but also the directions of the edges are subject to modification, or new edges may be generated or eliminated.

This leads to two fundamentally different approaches towards the Web: *Static approaches* describe the Web regarding a snapshot of a current state of the Web. *Dynamic approaches* consider the growth process of the Web.

After looking at the benefits, goals and reasons for regarding the Web graph, different models of the Web graph and their implications, problems and benefits will be discussed in this chapter. Afterwards, different approaches of storing the Web graph, accessing it, and considering its ongoing changes in the storing process will be shown.

2.3.1 Reasons for Regarding the Web Graph

There are several reasons and goals to analyze the Web's graph structure. Understanding this structure can help reaching the goals described in the following when searching the Web. Comparable reasons underlie the studies of Broder et al. (2000).

1. *Develop and evaluate crawling strategies.* Crawling strategies may maximize the amount of crawled webpages given certain resources, or gather a predefined set of webpages with a minimum resource usage, or look for certain communities on the Web.

2. *Establish a ranking function.* The Web structure is also used to calculate an importance measure for webpages. Hyperlinks pointing to a page can be regarded as votes for this page, where the votes of "important" webpages have a larger weight than the ones of "less important" webpages. An importance measure derived from this concept is used to sort search results. Different approaches will be shown in Section 2.4.

3. *Grouping webpages.* Under the assumption that strongly linked webpages are more probably dealing with a similar topic than webpages which are poorly connected among each other (or not linked at all), webpages can be grouped by identifying communities or clusters. Different approaches will be shown in Chapter 3.

4. *Identifying spam.* If the analysis of the Web structure identifies some kind of "natural" link structure this knowledge can help to identify also mainly

"artificially" linked groups of webpages. Spam clusters (e.g. link farms) often create thousands of linked webpages without any significant content, for the only purpose of attracting users (and potential customers) and especially search engines to their sites. Those search engines that are able to identify the special structure of spam will also be able to filter this spam and improve result quality in this way. Different approaches will be shown in Subsection 4.4.1.

A theoretical foundation can support the reaching of the goals 1 – 4. Understanding the structure of the Web and its development in the past helps to make predictions about the future development of the Web. The research performed in this area will be discussed in the following subsections.

2.3.2 Properties of the Web Graph

The Web graph's properties can be considered from different perspectives and used for different purposes. *Static approaches* describe the Web regarding a snapshot of a current state of the Web, whereas *dynamic approaches* try to explain the generative mechanisms leading to the creation of the Web graph and try to predict its future.

Depending on the research purpose, different entities representing Web elements might be represented by vertices. The most intuitive way may be to regard a graph where webpages act as vertices. Assuming that a vertex should represent an atomic entity of content, this can also be a frame on the page, a paragraph or (as smallest linkable entity) the anchor text or the graphic connected to the link. The anchor text is often used by search engines as an independent context descriptor for the webpage linked to. It can be considered as independent if the link's target belongs to the website of another host or provider. This decision can be supported by a graph consisting of websites as vertices and by analyzing the links of a subgraph containing this site. Another way of creating a Web graph is looking at single IP addresses or IP groups of the hosting Web servers as vertices. IPs are also used by search engines to identify false or artificial links that are created just for the reason of influencing result-page positions (spam).

The edges of the Web graph are formed by hyperlinks between the above enumerated entities – either webpages or websites. Depending on the purpose, they can be seen as directed or undirected edges. Weights can be assigned to the edges relative to the age of the link in order to obtain a graph that is more stable than the real Web and to overcome problems that occur with respect to the very dynamic graph. Another way to achieve a more stable graph is to gather only a small snap-

shot of a subgraph of the Web. Even in this case, one can never fetch all pages at exactly the same point in time. Thus, the resulting subgraph will not be a well defined graph. In order to get a clear model, the static approach regards a slowly changing subgraph of the Web.

Bharat et al. (2001) see the Web as a hierarchically nested graph with domains, hosts and websites as intermediate levels of affiliation. They studied the Web connectivity on a *host graph* with hosts as vertices and weighted edges representing the number of hyperlinks between pages on the corresponding host. Analyzing the relatedness of hosts by link frequency, they identified several explanations for strongly connected host pairs that did not seem to be related for any other reasons as the following:

1. *Large hosts:* Hosts like `www.geocities.com` and `members.aol.com` have a large number of connections because of their immense size.

2. *Boilerplate:* Hosts using a page template on all pages lead to many cross-host links (e.g. mirrors of the Open Directory Project tend to point to `www.dmoz.org` on every page).

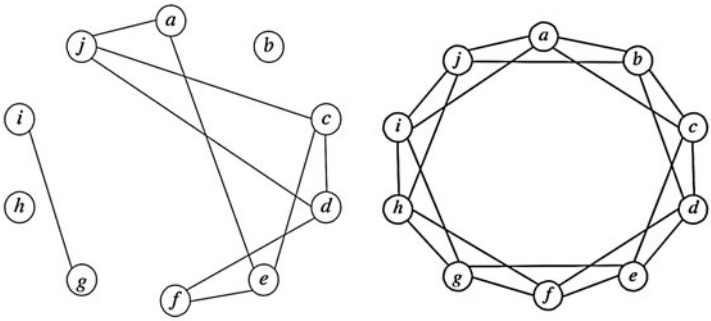
3. *Multi-host sites:* A site that spans multiple hosts may have many references between the hosts.

4. *Spam:* Search engine optimizers try to build up highly connected graphs to promote specific websites.

5. *Affiliate programs:* Website owners encourage third party websites to put links back to their sites rewarding them for the traffic sent through (like `www.amazon.com`).

A directed graph with vertices corresponding to static webpages and edges corresponding to links between the pages was studied in deep by Broder et al. (2000). They have analyzed the graph's properties like its diameter, degree distributions, connected components and its macroscopic structure. Experiments based on a Web crawl of 200 million pages and 1.5 billion links where conducted to verify that the degree distributions in terms of links on a page follow a power law. Further analysis of the connectivity among webpages results into a classification into differently connected components of the Web. Pennock et al. (2002) have analyzed distributions of links among subcategories of pages.

In the following, a selection of Web graph models is described in more detail. After the explanation of *random graph models*, an analysis of the connectivity of the Web is presented and some dynamic models are introduced.



(a) Erdös-Rényi random graph with 10 vertices and 9 edges

(b) A ring lattice (n=10, d=4)

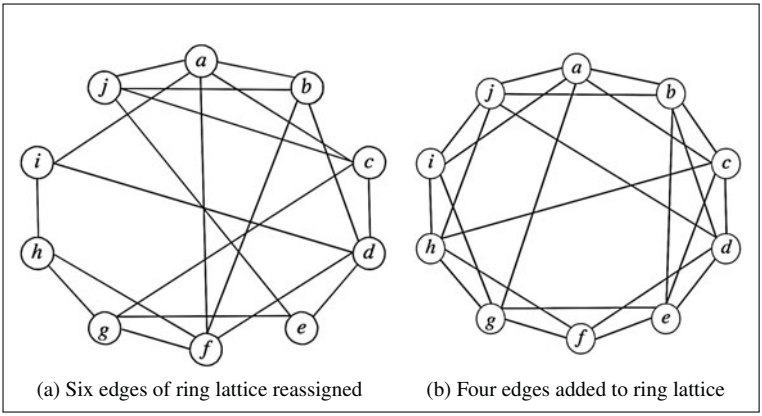
Figure 2.2: Random-graph examples I (Deo and Gupta, 2001b)

Random Graph Models

Deo and Gupta (2001b) use *random-graph models* for the description of the Web. The Erdös and Rényi Model (Erdös and Rényi, 1960) was the earliest model of a random graph. It explains the structure of a randomly created graph starting with n isolated vertices where each of the $\frac{n(n-1)}{2}$ pairs is connected with an edge with the same probability p (compare example in Figure 2.2a). This static model has two major disadvantages. It does not take into account the increasing number of vertices in the Web, and the edge formation is based on a non-uniform probability.

More suitable for describing the Web are *small-world networks* which were first described by Milgram (1967) in a social sciences context. Small-world networks are characterized by sparseness, small diameter and cliquishness. There are two variations of small-world models: the edge-reassigning and the edge-addition small-world network.

In the *edge-reassigning model* (Watts and Strogatz, 1998), the creation of a network starts with a ring lattice where each vertex is connected to its d nearest neighbors (see Figure 2.2b). In a reassignment step, each of the $\frac{n \cdot d}{2}$ edges is randomly removed and added to another vertex with a probability $0 \leq \sigma \leq 1$ (compare example in Figure 2.3a). The value of σ determines the evolution of the network: Setting $\sigma = 0$ will not change the graph at all. Increasing σ will decrease the shortest paths between two vertices in the network rapidly. A value of $\sigma = 1$ will lead to the same results as the Erdös-Rényi model.



(a) Six edges of ring lattice reassigned (b) Four edges added to ring lattice

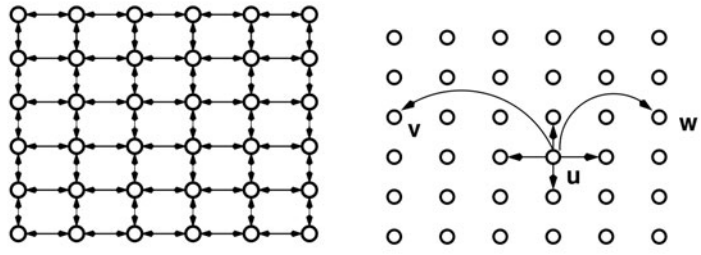
Figure 2.3: Random-graph examples II (Deo and Gupta, 2001b)

In the *edge-addition small-world network* (Newman et al., 2000), $\frac{\sigma \cdot n \cdot d}{2}$ new edges are added randomly to an existing ring lattice. For this purpose, two vertices are randomly selected and connected by the new edge. σ denotes again a new-edge-probability, d the degree of each vertex in the original ring-lattice, and n the number of vertices (compare example in Figure 2.3b).

Both models are characterized by a small diameter of the resulting graph. That means in the Web that the number of clicks between two pages on the Web is relatively small. Kleinberg (2000) uses this phenomenon to create algorithms for navigation in these networks. These networks are built with many local connections and few long-range connections following the paradigm of Watts and Strogatz. Instead of starting with a ring, a two-dimensional grid is used as basic structure (Figure 2.4a). Edges are allowed to be directed. Based on a distance measure using steps on the lattice, each vertex has connections to local neighbors and – with a certain probability – to distant vertices (see Figure 2.4b). The geographic interpretation is simple. Individuals have some local neighbors as well as a certain number of acquaintances distributed more broadly across the grid. In this model, routing messages from one point to any other is feasible using local information, if there is a correlation between the local structure and long-range connections.

Connectivity of the Web

During several experiments, Kleinberg et al. (1999) studied some of the local properties of the Web graph. Traditional random-graph models could not well explain



(a) A two-dimensional grid network with local neighbors of one step distance

(b) Node u with four local contacts and two long-range contacts v and w

Figure 2.4: Grid network model of Kleinberg (2000)

their observations. The number of links pointing to a webpage defines the in-degree of the corresponding vertex, and the number of links coming from a page the out-degree, respectively. Kleinberg et al. found that the probability for a vertex having an in-degree i is proportional to a power-law distribution $1/i^{\alpha}$ (with $\alpha \approx 2$). A random-graph model does not suitably describe the creation of this distribution, because its in-degrees exhibit either a Poisson or a binomial distribution. The same is true for the out-degree, because it also follows a power-law distribution.

Broder et al. (2000) describe a slightly higher exponent $\alpha \approx 2.72$ for the out-degree. The exponents turned out to be stable over different crawls. Moreover, they searched for *connected components* in the Web graph. A *component* of an undirected graph is defined as a set of vertices in which for any pair of vertices u and v in the set, there is a path from u to v. The undirected graph is created by ignoring the directions of links in the directed graph.

The components obtained in such manner are referred to as *weakly connected components* of the directed graph. In a crawl of approximately 203 million vertices, Broder et al. (2000) identified a weakly connected component of 184 million vertices which correspondents to a proportion of over 90 %.

Looking at the directed graph, this component can be broken down into four major pieces. A *strongly connected component (SCC)* of a directed graph is a set of vertices in which for any pair of vertices u and v in the set there is a directed path from u to v. The strongly connected component made up 56 million vertices in the above mentioned crawl (see Figure 2.5). Another piece consists of pages that can reach the SCC, but cannot be reached from it. It possibly consists of new sites that people have not yet discovered and linked to. This piece called *IN* consists of 44

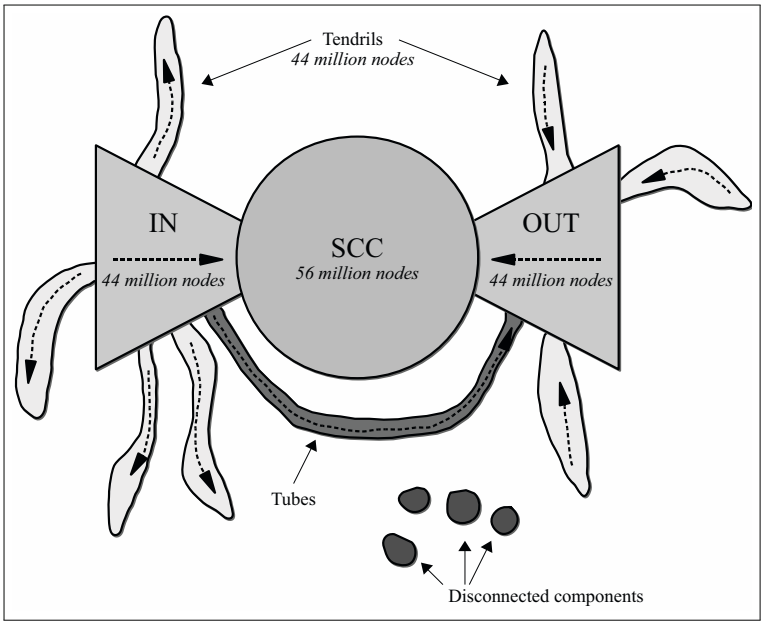


Figure 2.5: Connectivity of the Web – the "bow tie" structure (Broder et al., 2000)

million vertices. Pages that are accessible from the SCC, but do not link back to it, are part of the piece *OUT*, which also consists of 44 million vertices. These can be e.g. corporate websites that contain only internal links. The last piece called TENDRILS contain pages that can neither reach the SCC nor can be reached from the SCC.

The four pieces of the Web graph where put together in the *"bow tie"* graph shown in Figure 2.5, named after the shape of a bow tie. About 16 million vertices are not connected with the weak component at all.

Dynamic Models

The models described above are static and do not take into account the fact that new webpages are created, that new links are created, and links and pages are deleted. The *preferential-attachment model* (Barabási and Albert, 1999) reproduces one important finding of the Web. The probability that a page or a vertex i has degree d_i follows a power law. The exponent of the power law distribution was empirically found to be $\gamma = 2.9 \pm 0.1$, independent of time and number of edges. Barabási et al. (2000) analytically derived this exponent and found it to be $\gamma = 3$.

This model is criticized because it only adds edges to newly created vertices. But reconnections of existing edges or new connections between old vertices are also being established in the Web. Huberman and Adamic (1999) try to overcome these problems with their *website-growth model,* where the number of pages added to a site at a given time is considered proportional to those already existing on the site.

2.3.3 Representation and Storage of the Web Graph

With respect to the data forming the Web, one can distinguish between the Web's content data and its link data. The storage of content and its particular requirements were discussed in Section 1.3. The graph structure is built up by hyperlinks, and thus other, additional problems occur during its storage process. As with the textual storage of content, the amount of data consisting of link relations can grow very large. Consequently, access speed also becomes a critical issue during the specification of data structures.

A main difference to the search in textual content results from the fact that one might want to traverse the graph's vertices in both directions, even if the underlying graph is directed. Using link analysis methods, a quite common question concerning a webpage is which other pages point to this webpage. Information of hyperlinks pointing from a page a to a page b can be found only in the HTML code of page a. Knowing the code of page a, it is trivial to find all links pointing *from it.* These links are called *outlinks* or *successors* of a. But there is no information in the code of page a about the pages that point *to* page a. Consequently, one needs to know and to analyze the whole graph in order to find those hyperlinks pointing to page a. These hyperlinks are also called *inlinks* or *predecessors* of a. Moreover, one needs to store this graph in a structure that allows for this kind of request.

Connectivity Server

To resolve the above described problems, Bharat et al. (1998) have designed appropriate data structures, which are described in the following. They implemented these structures and introduced real Web data. The result was put into practice creating the so-called *connectivity server*. Its system architecture allows a fast navigation on the Web graph via the predecessor/successor relation. The server accepts queries consisting of a set of one or more URLs and returns a list of all incoming and outgoing links of this set.

In order to define appropriate data structures, webpages will be represented by vertices in the graph and hyperlinks on page a pointing to page b as directed edges between the vertices a and b. The set of vertices is stored in an array (node table), and for each vertex, an adjacency list containing its successors is maintained (see

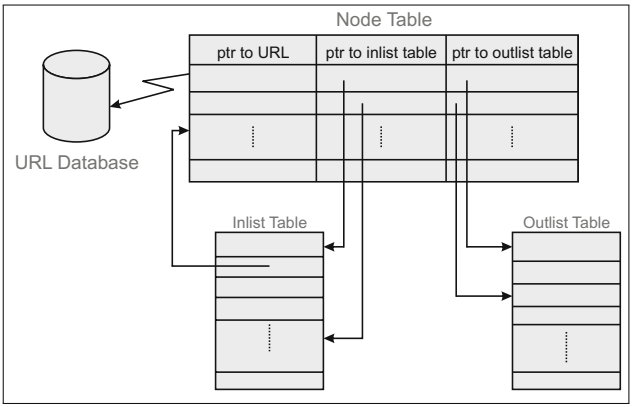


Figure 2.6: The Connectivity Server: Representation of the Web graph (Bharat et al., 1998)

Figure 2.6). An inverted adjacency list containing the predecessors is maintained for each vertex, too. The successor lists are stored in a common outlist table with pointers from the node table, and the predecessor lists are stored in a common inlist table.

The URL for each vertex or webpage is stored lexicographically sorted in a separate URL database. Since the common prefix between two URLs from the same server is often quite long, only the difference between the current and the previous URL is stored. This results in a size reduction of about 70 %.

The connectivity server performs three steps in order to process queries. It translates the query URLs to vertex IDs, returns the Web graph information around these vertices, and translates the resulting IDs back to URLs. The processing time on a 300 MHz Digital Alpha with 4 GB memory was approximately 0.1 ms per URL in the result set, working on a crawl of size of 100 million URLs. The third step takes most of the processing time, so applications working with internal IDs only can expect faster processing times.

Apart from a direct query interface a visualization of connectivity data in the neighborhood graph is being delivered by the server. Further applications where a fast access to link structures is needed will be described in Section 2.4. Compression techniques for reducing the database size are introduced by Suel and Yuan (2001) and Boldi and Vigna (2004).

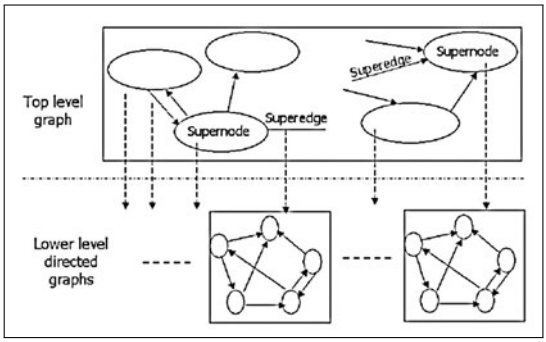


Figure 2.7: S-Node representation – two-level representation of a Web graph (Raghavan and Garcia-Molina, 2003)

Supernode Representation

Raghavan and Garcia-Molina (2003) point out the performance issues of a naïve graph representation. A lack of a schema to describe the structure of Web graphs can significantly increase query execution times and thus limit the utility of Web repositories. They propose the *Supernode (S-Node)* representation of the Web graph, which is highly space-efficient, and therefore enables in-memory processing of large Web graphs.

This representation also allows for queries that exceed the capabilities typically provided by commercial search engines. Their databases are mainly created for the purpose of responding simple keyword or phrase queries. Even their "expert" interfaces provide only simple extensions like limiting results to a specific domain. More sophisticated queries to the Web graph (such as "generate a list of universities that refer to Stanford researchers working with mobile devices") are not supported by commercial search engines. Without efficient data structures and access methods, search engines are not capable of answering complex graph queries in acceptable time.

The proposed more complex query systems provide users with the ability to use three different views on the Web repository at the same time:

1. a collection of text documents which can be searched, ranked, and classified using keywords or phrases,

2. a huge navigable directed graph with vertices representing pages and directed edges representing hyperlinks, and

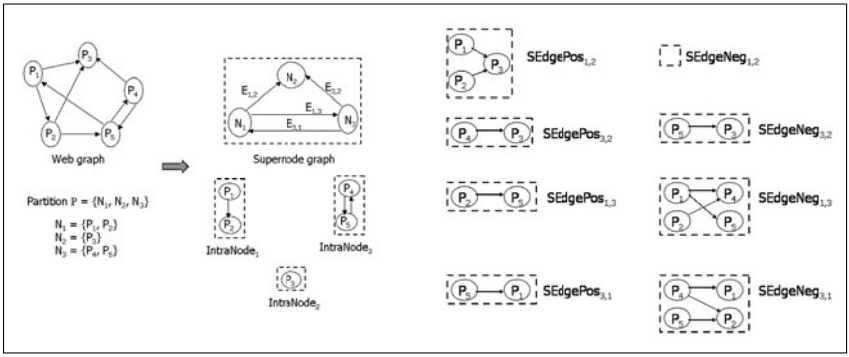


Figure 2.8: Partitioning the Web graph – S-Node representation example (Raghavan and Garcia-Molina, 2003)

3. a set of relational tables storing properties of webpages (like PageRank, length, title, and domain) on which relational queries can be applied.

The design process of a system that efficiently answers the queries described above leads to several challenges. The size and the growth of the Web lead to corresponding sizes of Web repositories, so that even relatively small repositories often consist of more than a hundred million webpages. Web graphs do not belong to a family of graphs for which efficient storage structures have yet been developed in graph-compression literature. Consequently, a direct adaption of known procedures is not possible.

The S-Node representation helps to face these challenges and to access a Web repository using the above mentioned three views simultaneously. The Web graph's nodes and edges are partitioned into two levels of graphs (see Figure 2.7). On the lower level, several smaller directed graphs are created, each of them containing only the interconnection within a small subset of webpages. On the top level, a directed graph is built using low-level graphs as nodes (so-called *supernodes*). Supernodes are connected with *superedges* containing link information between the low level graphs. Observed properties of the Web graph (see Broder et al., 2000; Kumar et al., 2000) are used to conduct the grouping of pages into supernodes and the compressing of the lower-level directed graphs.

Note that the S-Node representation does not drop any information belonging to the original graph. The structure will be defined as follows.

The directed graph of the Web will be denoted with $G = (V,E)$. $V(G)$ refers to the vertex set of graph G, and $E(G)$ to its edge set. A graph vertex representing a

page P will also be denoted by P. For a partition $\mathscr{P} = \{N_1, N_2, \ldots, N_n\}$ of $V(G)$ ($N_i \subseteq V(G)\ \forall i$ and $N_i \cap N_j = \emptyset\, \forall i \neq j$), several directed graphs are defined: a *supernode graph, n intranode graphs, n positive superedge graphs* and *n negative superedge graph*s. The different types of graphs are explained in the following using the simple Web graph example in Figure 2.8, which consists of five webpages $P_1, \ldots, P_5$ that are connected with eight directed links.

The supernode graph contains for each of the n partitions N_i a vertex called *supernode*. The five webpages from the example are assigned to one of three partitions, e.g. pages P_1 and P_2 form the partition N_1. The directed edges between two supernodes are called superedges. They are created using the following rule: a directed superedge $E_{i,j}$ is created from N_i to N_j if there is *at least one* page in N_i that points to *some* page in N_j. In the example, the supernode graph contains a superedge $E_{1,2}$ because $P_2 \in N_1$ points to $P_3 \in N_2$. Four superedges exist: $E_{1,2}$, $E_{1,3}$, $E_{3,1}$, and $E_{3,2}$. A superedge $E_{2,j}$ does not exist for any j, because N_2 is only consisting of P_3, which does not have any outlinks.

The interconnections between pages belonging to the same partition N_i are stored in an intranode graph $IntraNode_i$. In partition N_1 of the example, $Intranode_1$ consists of the hyperlinks between page P_1 and page P_2.

For the links between pages of different partitions, two types of superedge graphs are defined. Both types can be alternatively used. A *positive* superedge graph $SEdgePos_{i,j}$ is a bipartite graph containing those links that point from pages in N_i to pages in N_j. In the example of Figure 2.8, $SEdgePos_{1,2}$ contains two edges representing the two links from $P_1 \in N_1$ and $P_2 \in N_1$ pointing to $P_3 \in N_2$. $SEdgePos_{i,j}$ is only defined, if there exists a corresponding superedge $E_{i,j}$. A *negative* superedge graph $SEdgeNeg_{i,j}$ is also defined as a directed bipartite graph. It contains out of all possible links that may point from pages in N_i to pages in N_j those links that *do not exist* in the actual Web graph. In the example, $SEdgeNeg_{3,2}$ contains the link from $P_5 \in N_3$ to $P_3 \in N_2$, because out of the two possible links, the link from $P_4 \in N_3$ to $P_3 \in N_2$ exists in the original Web graph. Since all pages of N_1 point to all pages of N_2, $SEdgeNeg_{1,2}$ does not contain any edges.

For each partition $\mathscr{P}$ on the vertex set $V(G)$ of G, an S-Node representation of G, denoted as $SNode(G, \mathscr{P})$, can be constructed using a supernode graph that points to a set of intranode graphs and to a set of positive or negative superedge graphs. For memory-economic reasons, either the corresponding positive superedge graph or the corresponding negative superedge graph is stored for each superedge $E_{i,j}$, depending on which of the two superedge graphs has the smaller number of edges. The resulting representation of the example is shown in Figure 2.9. As

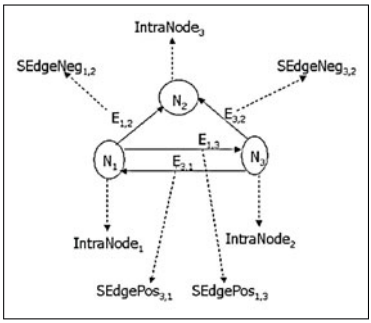


Figure 2.9: S-Node representation of a Web graph example (Raghavan and Garcia-Molina, 2003)

$SEdgeNeg_{1,2}$ has fewer edges than $SEdgePos_{1,2}$, the superedge $E_{1,2}$ points to the negative superedge graph.

The partition of the Web graph must produce intranode and superedge graphs that are highly compressible, and it must enable queries to be executed only on a relatively small number of intranode and superedge graphs. Therefore, the generation of the partition makes use of some observations about Web graphs. Often, new pages on the Web add links by copying links from an existing page. These pages have very similar adjacency lists. In this case, a significant number of links on a page point to other pages from the same domain. Pages with similar adjacency lists are likely to be related to each other. These properties of the Web graph lead to the postulation of the following properties of a well suited partition:

1. Pages with similar adjacency lists are tried to be grouped together.

2. All pages of each partition element belong to the same domain.

3. Out of pages belonging to the same host, those with lexicographically similar URLs are more likely to be grouped together.

Compared to other schemes for representing Web graphs (see Adler and Mitzenmacher, 2001; Randall et al., 2002), the S-Node representation reaches a higher compression and therefore, it scales better, especially if the graph fits in the memory. This leads to a significant reduction in query execution time. It proves to be 10 to 15 times faster than other schemes. The S-Node representation can also serve as a high-performing data structure for the methods described in Section 2.4.

2.4 Link Analysis

Search engines that purely consider textual information when creating their results face several problems. If they, for example, measure the relevance of a webpage based on keyword frequencies, they can easily be tricked by webpages stuffed with these keywords. In the opposite case, webpages may not contain certain keywords, even if they well describe their contents.

By means of link analysis, additional information can be extracted from the Web graph to overcome these problems. This section describes several methods of link analysis.

2.4.1 Ranking

Link analysis has become an important means for ranking search engine results. An importance measure can be determined by counting the inlinks of a webpage and weighting them again with the importance of the linking site. The underlying hypothesis is that a hyperlink can be seen as a vote from one webpage to another. In this context, a vote from a more important webpage weighs more than a vote from a less relevant site. The importance weight of a webpage is then shared over the webpages it votes for. This leads to a simple recursive definition of the PageRank r_u of a webpage u (Page et al., 1998):

$$r_u = c \sum_{v \in B_u} \frac{r_v}{N_v},$$

where the PageRank values r_v of all webpages $v \in B_u$ that contain hyperlinks pointing towards u (backlinks) is divided by the number of links N_v the webpage v contains. A factor $c \leq 1$ is introduced for normalization so that the total rank for all webpages stays constant. The factor c is smaller than one because there are pages that do not contain any outgoing links. The recursive equation may be computed by starting with any set of ranks and iterating the computation until it converges. Arasu et al. (2001b) have shown the convergence of the computation.

Figure 2.10a demonstrates the PageRank of a simple graph using a small example of five webpages (Arasu et al., 2001a). For the sake of simplicity, the factor c is set to $c = 1$. Webpage 2 distributes its PageRank of $r_2 = 0.286$ via its outgoing links to webpage 1 and 3. Each webpage receives half of the rank $\left(\frac{r_2}{2} = 0.143\right)$ because page 2 possesses two links. Since page 3 does not have any other incoming

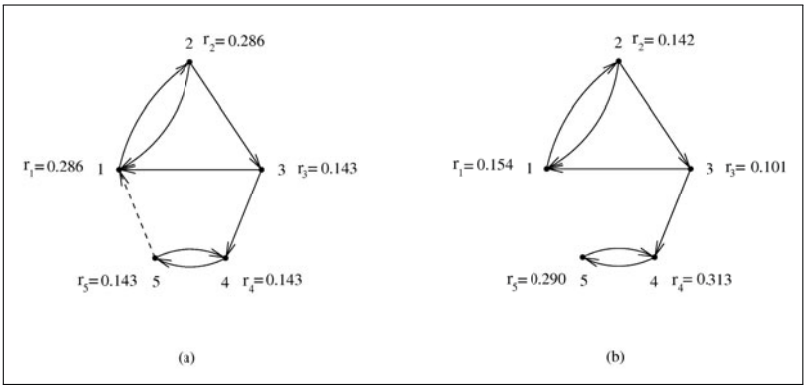


Figure 2.10: PageRank calculation examples: (a) Simple PageRank (b) Modified PageRank with $c = 0.8$ (Arasu et al., 2001a)

links, its rank becomes $r_3 = 0.143$. The rank of page 1 calculates as

$$r_1 = \frac{r_2}{2} + \frac{r_3}{2} + \frac{r_5}{2} = 0.143 + 0,0715 + 0,0715 = 0,286.$$

The rank of page 1 is relatively high because page 1 has three incoming links. In the simple model, page 2 has the same rank as page 1, because it receives all the rank from page 1. The sum of all ranks is $\sum_u r_u = 1$.

Another notation and calculation of the PageRank is starting with a square matrix A with rows and columns corresponding to webpages. $A_{u,v} = 1/N_u$ if a link is pointing from u to v. Otherwise it is 0. Letting R be a PageRank vector over webpages, then $cR = AR$. R is an eigenvector of A with eigenvalue $c > 0$. An eigenvector of A may be computed by repeatedly applying A to any non-degenerate start vector.

In this simplified model, a problem occurs if for example two pages point to each other but have no outgoing links. This will happen after removing the dotted arrow in Figure 2.10a between page 5 and 1. If another webpage points to one of these pages (in this case, page 3 points to page 4), the loop consisting of 5 and 4 will accumulate rank during the iteration but will never distribute any rank to other pages. For this reason, this kind of loop is called a *rank sink*.

The sink problem can be faced by assigning the decay factor c to a value $0 < c < 1$, that means assigning only the fraction c of the PageRank to the nodes, and distributing the remaining rank among all the pages on the Web. This leads to the

modified PageRank definition r'_u:

$$r'_u = c \sum_{v \in B_u} \frac{r'_v}{N_v} + \frac{(1-c)}{m},$$

where m is the total number of nodes in the graph. The sample in Figure 2.10b shows the graph of Figure 2.10a without the link $1 \rightarrow 5$ and ranks calculated for $c = 0.8$. Pages 5 and 4 now have higher ranks, because they "collect" rank which they do not give away.

Another, quite intuitive interpretation of PageRank leads to the *random surfer model*. Corresponding to the simple PageRank definition, the model bases on the assumption that there is a person randomly surfing the Web, that means he or she clicks on successive links of visited webpages at random. The probability of being on a certain page corresponds to the PageRank vector R. It follows the rules of a random walk (Motwani and Raghavan, 1995) on the Web graph.

In the simple model (Figure 2.10a), the random surfer can get stuck in the loop between the pages 4 and 5. The additional summand in the modified model (Figure 2.10b) expresses that the surfer periodically gets "bored" and jumps to a random page on the Web instead of following a link. The factor c specifies the frequency the surfer gets bored.

The calculation of PageRank using power iteration is described by Arasu et al. (2001b). They have compared different methods and graph structures, under which the convergence of iterations works faster than computing the eigenvector.

Another approach for computing ranks is also based on the random surfer model. The *Page Reputation* algorithm proposed by Rafiei and Mendelzon (2000) extends the PageRank algorithm by considering the search term τ in the calculation. The reputation of a page u is defined as the probability that a random surfer who looks for the topic τ visits page u.

Let $|U_\tau|$ be the total number of pages on the Web containing the term τ. The probability that the surfer visits page u in a random jump is

$$R^0(u,\tau) = \begin{cases} \frac{c}{|U_\tau|} & \text{if term } \tau \text{ appears on page } u \\ 0 & \text{otherwise.} \end{cases}$$

If one considers that the surfer follows n links before he reaches a certain page, the reputation can be calculated in an iterative process. Let N_u be the number of outgoing links of page u. Intuitively, the probability that a surfer visits page u after visiting page v through the link $v \rightarrow u$ is $\frac{1-c}{N_v} R^{n-1}(v,\tau)$ where $R^{n-1}(v,\tau)$ is the probability that the surfer visits page v for topic τ.

Thus, the probability $R^n(u,\tau)$ of visiting page u for topic τ after n steps can be determined as follows:

$$R^n(u,\tau) = \begin{cases} \frac{c}{|U_\tau|} + (1-c)\sum_{v \in B_u} \frac{R^{n-1}(u,\tau)}{N_v} & \text{if term } \tau \text{ appears on page } u \\ (1-c)\sum_{v \in B_u} \frac{R^{n-1}(u,\tau)}{N_v} & \text{otherwise.} \end{cases}$$

When increasing the number of iterations during the calculation of probabilities $R^n(u,\tau)$, the reputation rank $\pi_{u,\tau}$ of a page u on a topic τ is defined as

$$\pi_{u,\tau} = \lim_{n \to \infty} R^n(u,\tau).$$

Rafiei and Mendelzon prove that the notion of reputation rank is well-defined, i.e. for every term τ and every parameter $0 < c < 1$, there is a unique probability distribution $\pi_{u,\tau}$, provided that every page has at least one outgoing link. They calculate the matrix R where a row corresponds to a webpage and a column to each term that appears in the webpage containing the reputation values using an iterative process and show the convergence of the algorithm.

Zhang and Dong (2000) extend the random surfer model with more parameters and let the random surfer follow a *tendency matrix* when jumping from page to page. Let the relevance $\omega \cdot \mathrm{sim}(S_u, q)$ measure the similarity of the content of webpage S_u and the user's query q and let the authority μ be a measure of references made to the webpage. Further, let the integrativity θ measure the references made in the webpage and let the novelty metric ε measure the difference of a webpage from other pages. At each time of the random walk, the random surfer viewing a webpage S_u from the set of pages returned by a search engine at time t has four choices: staying on pages $S_{u,}$, clicking on a link v, going back, or selecting another webpage from the results.

The tendency matrix W showing his possible behavior is represented as

$$W_{uv} = \begin{cases} \omega \cdot \mathrm{sim}(S_u, q) & \text{if } u = v \\ \mu & \text{if } v \in N_u \\ \theta & \text{if } u \in N_v \\ \varepsilon & \text{otherwise,} \end{cases}$$

with the relevance $\mathrm{sim}(S_u, q)$ of result S_u for the query q and $0 < \omega, \mu, \theta, \varepsilon < 1$ and $\omega + \mu + \theta + \varepsilon = 1$.

The normalized tendency matrix results in a probability matrix T for the set of webpages S. A holomorphic and homogeneous Markov chain with the transition

matrix T converges in a unique distribution, which can then be used as a ranking criterion.

Agarwal and Chakrabarti (2007) have analyzed and enhanced algorithms for random walks that are used for ranking vertices in graphs. A link-based ranking scheme for search engines that are focused on a certain topic is presented by Abou-Assaleh et al. (2007).

2.4.2 Authorities and Hubs

Beyond the ranking of search results, link based techniques can help to identify webpages that serve as *authorities* or *hubs* for a certain topic in the Web. Kleinberg (1999) introduced the notion of authority in relation to a broad-topic query. The question raised for a particular webpage is how to determine whether this webpage is authoritative in the context of the topic regarded. An appropriate model is supposed to filter the most authoritative pages out of a collection of relevant pages.

Such a model quickly reaches the limits of purely text-based analysis. Assuming for example that one of the most authoritative sources for the query "Harvard" will be the homepage `www.harvard.edu` of the Harvard University. Looking at the millions of pages this query returns, most of them contain the word "harvard" and there is no purely endogenous measure based on information contained in each page that can lead to a decision about its authority. Among the pages found, `www.harvard.edu` is neither the one that uses the term "Harvard" most often nor gives it any other valid hints, why this page should be more authoritative than other pages.

The problem for text-based search engines grows if the search term does not occur at all on natural authority sites. E.g., the query term "search engine" should lead to some major search engines, but many of them do not use this term on their pages. The same is true for big automobile brands, who do not write the term "automobile manufacturer" on their website.

Analyzing the hyperlink structure can help to overcome the above mentioned problems. Like in the context of PageRank, a hyperlink is interpreted as a considerable amount of latent human judgment. The creator of a page a linking to a page b has conferred authority to page b. Measuring the authority by regarding inlinks helps to solve the problem that many pages are not self-descriptive.

As basis for the *HITS (Hyperlink Induced Topic Search)* algorithm a link-based model for the conferral of *authority* is proposed. It focuses on the relationship between the authorities for a topic and those pages that link to many related authorities. Latter pages are called *hubs.*

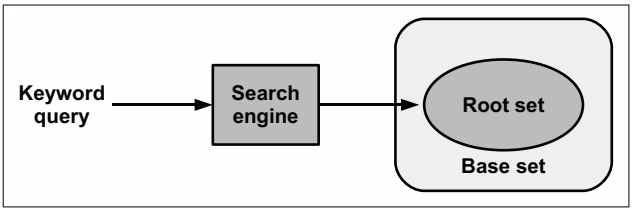


Figure 2.11: The HITS algorithm – base-set creation

The algorithm works on a focused subgraph of the Web that will be constructed from the results of a keyword query sent to a text-based search engine (see Figure 2.11). The *focused subgraph* consists of a *root set* and an expanded set *(base set)* of webpages related to the topic of the original query. After the construction of the focused subgraph, which is described in the following, its link structure will be analyzed to identify authorities and hubs.

Like earlier in this chapter, the Web will be regarded as a directed graph $G = (V,E)$ based on webpages (set of vertices V) and hyperlinks (set of edges E). $G[W]$ is a subgraph consisting of a subset of pages $W \subseteq V$ and its edges that correspond to all the links between the pages in W.

An ideal subset used as a focused collection of pages for further analysis should own the following properties:

1. The collection is relatively small. This helps to limit the computing times to a manageable amount.

2. The collection is rich in relevant pages. This makes it easier to find good authorities.

3. The collection contains most (or many) of the strongest authorities. This is a prerequisite to achieve good results.

For the creation of such a collection, Kleinberg (1999) proposes the process shown in Figure 2.11. Instead of using the whole result set Q_σ of a query σ, the following approach is chosen mainly for two reasons. First, the set Q_σ is in many cases two large to satisfy condition 1. Second, not all authorities do necessarily contain the query term σ, and for this reason, they may not be contained in Q_σ. Thus, the result set will be limited to t pages (typically $t := 200$). In a second step, the set will be expanded by pages linking from or to the result set.

The base-set generation process will be described by means of Algorithm 2.1. The CREATESUBGRAPH procedure starts with a keyword query σ sent to a text-

Algorithm 2.1 HITS algorithm: Base set generation (Kleinberg, 1999)

CREATESUBGRAPH($\sigma, \mathscr{E}, t, d$)
  σ: a query string
  $\mathscr{E}$: a text-based search engine
  t, d: natural numbers

let R_σ denote the top t results of $\mathscr{E}$ on σ
set $S_\sigma := R_\sigma$
for each page $p \in R_\sigma$ do
  let $\Gamma^+(p)$ denote the set of all pages p points to
  let $\Gamma^-(p)$ denote the set of all pages pointing to p
  add all pages in $\Gamma^+(p)$ to S_σ
if $|\Gamma^-(p)| \leq d$ then
  add all pages in $\Gamma^-(p)$ to S_σ
else
  add an arbitrary set of d pages from $\Gamma^-(p)$ to S_σ
end
return S_σ

based search engine $\mathscr{E}$. The top t results, building the root set R_σ, are stored as the initial base set S_σ. For every p in the root set, all pages in the set $\Gamma^+(p)$ page p links to (outlinks) are added to the base set S_σ. Figure 2.12 illustrates the relationship between the root set and the base set. $\Gamma^-(p)$ denotes those pages that link to page p (inlinks). A maximum number of d pages of $\Gamma^-(p)$ is added to the base set S_σ.

Constructing a base set using the CREATESUBGRAPH procedure with $t = 200$ and $d = 50$ typically satisfies the properties 1 – 3. and returns a base set in the range of 1000 – 5000 pages. In order to avoid links that are only created for navigational purposes on the same website, all links between pages in the same domain are deleted. The resulting subgraph will be denoted as G_σ.

After having created a small subgraph G_σ that is relatively focused on the query topic and contains relevant pages and strong authorities, the next step is to extract these authorities. Ranking the pages only by in-degree involves some significant problems. In the subgraph G_σ, there may exist strong authorities regarding the search topic as well as universally popular pages that are not necessarily authorities with respect to the topic. An analysis of those pages that point to authorities shows an overlap in the sets of these pages. Those pages that have links to multiple

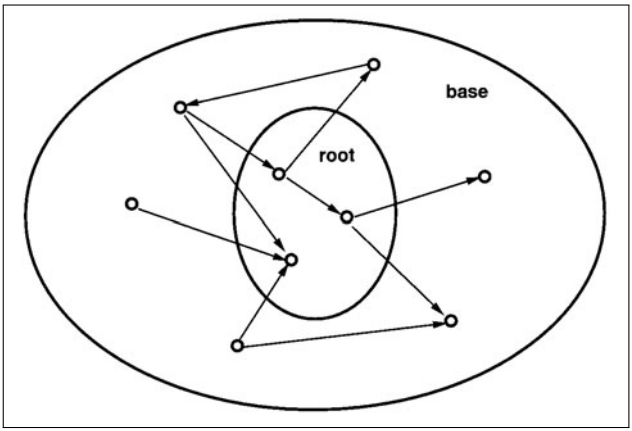


Figure 2.12: HITS algorithm – Expanding the root set into a base set (Kleinberg, 1999)

relevant authority pages are called *hub* pages. Hub pages group together authorities of a common topic and help to discriminate unrelated pages of large in-degree.

This leads to a definition of hubs and authorities based on their mutually reinforcing relationship:

1. A good hub is a page that points to many good authorities.
2. A good authority is a page that is pointed to by many good hubs.

The quality of authorities and hubs is expressed with weights assigned to the webpages in the subgraph G_σ. Each page p will be assigned an authority weight x^p and a hub weight y^p. Both are maintained in a normalized way so that $\sum_{p \in G_\sigma} (x^p)^2 = 1$ and $\sum_{p \in G_\sigma} (y^p)^2 = 1$. Thus, the authority weights are calculated as

$$x^p := \sum_{q|(q,p) \in E} y^q$$

(later referred to as $\mathscr{I}$ operation), and the hub weights are calculated as

$$y^p := \sum_{q|(p,q) \in E} x^q$$

(later referred to as $\mathscr{O}$ operation).

Both operations are performed in alternation during the ITERATE process (see Algorithm 2.2), storing the weights x^p in a vector x and y^p in a vector y. The

Algorithm 2.2 HITS algorithm: calculation of hubs and authority values (Kleinberg, 1999)

ITERATE(G,k)
  G: a collection of n linked pages
  k: a natural number

let z denote the vector $(1,1,1,\ldots,1) \in \mathbb{R}^n$
set $x_0 := z$
set $y_0 := z$
for $i := 1$ to k
  apply the $\mathscr{I}$ operation to (x_{i-1}, y_{i-1})
    obtaining new x-weights x'_i
  apply the $\mathscr{O}$ operation to (x'_{i-1}, y_{i-1})
    obtaining new x-weights y'_i
  normalize x'_i obtaining x_i
  normalize y'_i obtaining y_i
end
return(x_k, y_k)

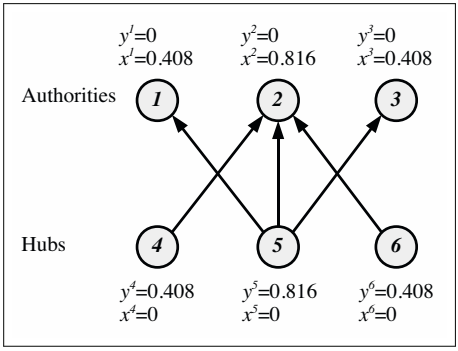


Figure 2.13: Authority and hub values of a simple graph

sequences of vectors x_k and y_k converge to fixed points x^* and y^*, respectively, for sufficiently large values of k. For a proof of convergence see Kleinberg (1999). Figure 2.13 shows the authority and hub values after the execution of the HITS algorithm on a simple graph.

The procedure can also be formulated in matrix notation as follows. Based on an $n \times n$ adjacency matrix A containing the values

$$a_{i,j} = \begin{cases} 1 & \text{if page } i \text{ points to page } j \\ 0 & \text{otherwise,} \end{cases}$$

the authority values written as vector $x = (x_1, x_2, \ldots, x_n)$ and the hub values written as vector $y = (y_1, y_2, \ldots, y_n)$ can be calculated as $x \leftarrow A^T y$ and $y \leftarrow Ax$.

This leads to the calculation of the authority vector

$$x \leftarrow A^T y \leftarrow A^T Ax = \left(A^T A\right) x$$

and the hub vector

$$y \leftarrow Ax \leftarrow AA^T y = \left(AA^T\right) y.$$

For any non-degenerate choice of the initial vector, the result of applying the power iteration technique to $A^T A$ converges to the principal eigenvector of $A^T A$ (Kleinberg et al., 1999).

The output of the HITS algorithm consists of a hub list and an authority list sorted by the respective weights. The first step (creation of the subgraph) uses textual search engines whereas the second step (iterative calculation of the values) totally ignores textual information. Anyway, the HITS algorithm returns good results for different types of queries.

The quality of the results depends not only on meeting the above described properties of the graph and a sufficient number of links in the graph. Bharat and Henzinger (1998) have discovered further conditions under which the algorithm does not come to satisfying results:

1. Mutually reinforcing relationships between hosts may distort the result, if for example a set of documents from one host points to a single document on a second host. The first host's contribution to the authority score does not reflect the real impact of this host. In the reverse case, the hub score can be influenced by a document pointing to multiple pages of a second host.

2. Automatically generated links of Web authoring tools do not reflect a human's "vote" for a page and may distort the results.

3. Non-relevant pages can be often found in the neighborhood graph. If there are different well-connected nodes belonging to another topic, they can lead to a topic drift. In this case, high-ranked authorities and hubs do not belong to the original topic.

The first problem can be solved by distributing the fractions of one vote from one host to one page of another host, i.e. weighting the links from k pages from one host to another with $\frac{1}{k}$. Problems 2 and 3 can be addressed by an additional content analysis of the neighborhood graph, leading to an improved algorithm. Chakrabarti et al. (1999a) address the limitations of HITS by assigning non-negative weights to each link, depending on the query terms and the endpoints of the links.

Further improvements of the HITS algorithms are described by Borodin et al. (2001). An evaluation of the effectiveness of HITS in comparison with other link-based ranking algorithms was performed by Najork et al. (2007). HITS outperformed PageRank in a large-scale study based on a crawl of 463 million webpages containing 17.6 billion hyperlinks and referencing 2.9 billion distinct URLs.

The combination of link-based and text retrieval algorithms delivers better results than PageRank in many contexts. Link-based features generally perform better for general queries, whereas text-based algorithms perform better for specific queries.

2.4.3 Other Link-based Methods

Besides the ranking algorithms and the authority/hub calculation, link analysis can serve for a variety of different Web mining applications. The following examples present link information as a valuable input for Web search and Web analysis methods.

Related Webpages

Dean and Henzinger (1999) use connectivity information in order to find related pages on the Web. Here, the search process is triggered by a URL, where traditional search engines receive user queries. The output is a set of related webpages, i.e. webpages that address the same topic as the original page.

Two algorithms, the *companion* algorithm and the *co-citation* algorithm, are purely based on linkage, neglecting the content and the usage of pages. The companion algorithm calculates hub and authority scores using the modified version of HITS (Kleinberg, 1999) based on a *vicinity graph* of the input URL u. In the definition of the vicinity graph, the terms *parent* and *child* are used for linked webpages as follows: if a hyperlink points from page w to page v, then w is a parent of v and v is a child of w. The vicinity graph of u consists of

1. the page u itself (the query URL),
2. up to B parents of u (“go back”), and for each parent up to BF of its children different from u (“back-forward”), and

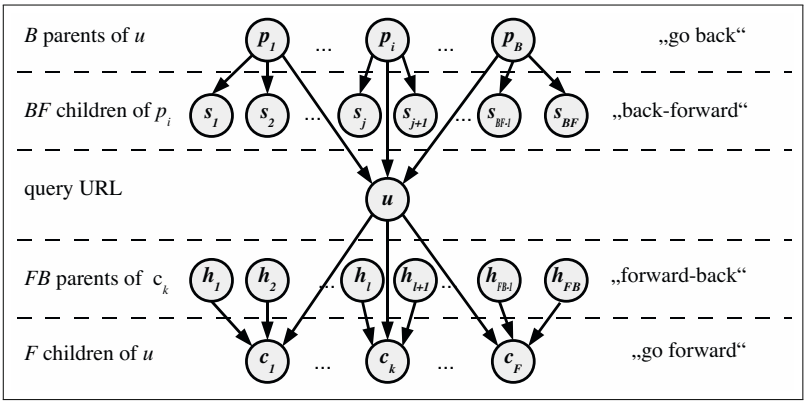


Figure 2.14: Vicinity graph of a page u

3. up to F children of u (“go forward”), and for each child up to FB of its parents different from u (“forward-back”).

After eliminating certain URLs which are obviously unrelated to most queries (one example is a link to `www.yahoo.com`), the vicinity graph is built selecting parents and children from the Web graph (compare Figure 2.14). If a parent p_i has too many children, e.g. more than $BF+1$ children, those $\frac{BF}{2}$ children preceding the link from p_i to page u and those $\frac{BF}{2}$ children succeeding the link from p_i to page u on page p_i are chosen.

In a next step, duplicates are eliminated, because many pages are duplicated across different hosts. Two pages are defined as near-duplicates if they have more than ten links and if they have at least 95 % of their links in common.

Edge weights are assigned following the improved model of Bharat and Henzinger (1998), described in 2.4.2. On the resulting weighted graph, the improved HITS algorithm is used to determine the highest authority scores (excluding u itself) as result.

A simple alternative approach for finding related pages is to use the co-citation algorithm. Two pages are co-cited if they are *siblings,* i.e. if they have a common parent. The degree of co-citation is determined by the number of common parents. The algorithm chooses up to B arbitrary parents of u. For the parent p_i, it adds up to BF children of p_i to a set S. The result of the algorithm consists of those pages of the set S that are most frequently co-cited with u.

In a user study, both algorithms outperform the precision of NETSCAPE browser’s “What’s related?” feature (Version 4.06) for finding related pages, even though

NETSCAPE takes into account user behavior as well as content data additionally to hyperlinks.

Hub Synthesis

The model of Achlioptas et al. (2001) uses three components of Web search in a combined approach: the link structure of the Web, the content generation process and the human searcher's query generation. They investigate the correlations between these components and present a Web search algorithm based on spectral techniques.

Related Hosts in the Host Graph

Bharat et al. (2001) regard the Web graph with vertices on different hierarchical levels. Domains, hosts and websites build intermediate levels of affiliation. They represent the connectivity between hosts by a directed graph, with hosts as vertices and weighted edges representing the number of hyperlinks between the pages of different hosts. Based on the host graph, they describe an algorithm which finds related hosts with high precision.

Categorization

Chakrabarti et al. (1998) expand text-classification algorithms by hyperlink information. They extract semantic information contained in hyperlink relations. Since link information is noisy, naïve use of terms in the link neighborhood of a document can even *degrade* accuracy. They propose statistical models and a classification technique by exploiting link information in a small neighborhood around documents. Their experimental results show that using hyperlink classifiers in addition to text classifiers significantly reduces the error that may emerge during the classification process.

Identification of Communities

The Web graph is also used to identify communities in the Web, i.e. groups of individuals who share a common interest (Kumar et al., 1999). *Explicitly-defined* communities are easy to find in appropriate newsgroups, portals, Web rings, or resource collections in directories. In addition to that, the chaotic nature of content creation in the Web has resulted in many more *implicitly-defined* communities. These communities often focus on a level of detail typically too fine to build large resource sites. Kumar et al. developed methods to identify implicit communities by analysis of the Web graph for three reasons:

1. Implicit communities can provide valuable information resources for a user,

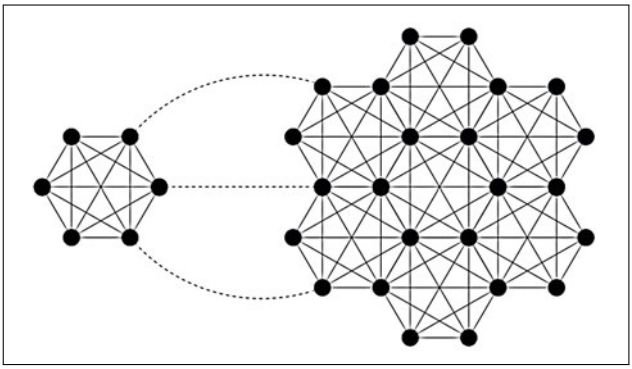


Figure 2.15: Community identification with maximum flow methods (Flake et al., 2002)

2. studying implicit communities gives insights into the evolution of the Web because they somewhat represent its sociology, and

3. distinguishing between these communities can target advertising at a very precise level.

Kumar et al. characterize communities by dense directed subgraphs of the Web. For the identification algorithm, the notion of *potential fans* is introduced. Fans are derived from specialized hubs created by e.g. a HITS algorithm. Based on in-degree and out-degree analysis of potential fans, appropriate candidates are selected and combined into non-overlapping cores. Experimental studies brought up several communities that were not explicitly described in the Web before.

Flake et al. (2002) use maximum flow algorithms to identify communities. They define a community as a collection of webpages built in such a manner that each member page has more links within the community than outside of the community. Maximum flow methods separate a graph into two subgraphs (see example in Figure 2.15) with any choice of a source vertex and a sink vertex. In the example, the three dashed links are removed to separate the left from the right subgraph. Flake et al. repeatedly apply a maximum flow procedure on a graph to identify multiple communities. Generally, the identification of a naturally formed community is intractable, because the basic procedures belong to the family of NP-complete graph-partitioning problems Garey and Johnson (1979). Assuming the existence of one or more seed websites, an algorithm based on the $s-t$ maximum flow problem (Ahuja et al., 1993) is proposed that efficiently identifies communities in a polynomial time.

Algorithm 2.3 Maximum flow community (Flake et al., 2002)

```
procedure MAXIMUM-FLOW-COMMUNITY
  input: graph G = (V,E); set S ⊂ V; integer k .

  Create artificial vertices, s and t and add to V.
  for all v ∈ S do
    Add (s,v) to E with c(s,v) ≡ ∞.
  end for
  for all (u,v) ∈ E do
    Set c(u,v) ≡ k.
    if (v,u) ∉ E then add (v,u) to E with c(v,u) ≡ k.
  end for
  for all v ∈ V; v ∉ S ∪ {s,t} do
    Add (v,t) to E with c(v,t) ≡ 1.
  end for
  call: MAX-FLOW(G,s,t).
  output: all v ∈ V still connected to s.
```

The $s-t$ maximum flow problem is defined on a directed graph $G=(V,E)$ with edge capacities $c(u,v) \in \mathbb{Z}^+$ and two vertices $s,t \in V$. Its goal is to find the maximum flow that can be routed from the source vertex s to the sink vertex t and that obeys all capacity constraints. The maximum flow of the network is the same as the flow through the minimum cut that separates s and t (Ford and Fulkerson, 1956).

The algorithm (compare Algorithm 2.3) interprets the hyperlinks as undirected edges. Starting with a set of seed webpages $S \subset V$, it identifies one community at a time applying the max-flow algorithm on the graph G.

Experimental results have identified communities of approximately 200 webpages, where the majority was found to be highly topically related.

Spam Identification

As already mentioned in the context of the algorithm of Dean and Henzinger, analyzing the link structure can also discover links that are artificially created for the only purpose of distorting results of search engines.

Spam in the context of static rank means, that webmaster are creating site clusters, which often consist of very similar sites, finally linking to one domain or document only. This kind of spam cluster can consist of millions of sites, not con-

taining any valuable content. This way, the static rank is becoming more and more a measure of the marketing budget of a site or the cleverness of the webmaster of a domain, rather than a measure of "real" reputation or content quality. As a result of this development, the importance of the static rank as a tool for determining the quality or the relevancy of a site is decreasing. A classification of different types of spam can be found in Henzinger et al. (2002).

If a link structure of a subgraph differs significantly from naturally created structures, this is an identifier for spam (see Subsection 4.4.1). In the following, an algorithm is proposed to identify the kind of spamming that targets the static rank. The basic idea of the static rank is reasonable – the more important sites refer (link) to a site, the more important it is. There is a way to discriminate between "natural grown" link clusters and "artificial" ones (spam).

In order to find a quantitative method that can discriminate between these two types of link clusters, the statistical distribution of the relevancy R_j^s is analyzed (for the definition of R_j^s see Section 1.4). A function is introduced that describes the statistical distribution of the relevancy R_j^s of the documents d_j pointing to the document d_i:

$$\phi(R_j^s) = e^{-\frac{\left(R_j^s - R_0^s\right)^2}{\sigma^2}},$$

where R_0^s is the average static rank of all sites, linking to the center of this cluster d_i. The parameter σ defines the standard deviation of the distribution.

The above mentioned types of clusters can be discriminated, using the distribution ϕ – natural grown clusters contain links from an inhomogeneous set of sites, for example, the links to a site of a well known university will come from very small (amateur) sites of students, employees and alumni (with a low PageRank), via semi professional institutional sites (spin offs, research partners, ...) up to sites of other highly ranked universities or institutes. The artificial link cluster consists of automatically generated sites, each of them usually optimized for different keywords, but having approximately the same static rank. As a result of this it is possible to introduce a "cut off" criteria based on the above distribution function. A cluster is most likely spam, if the condition

$$\sigma_{spam} < \sigma_{critical} \tag{2.1}$$

is fulfilled. Here $\sigma_{critical}$ is an empirical parameter, which can be determined from the analysis of known natural and artificial clusters (or from the software, generating the sites of the spam cluster). Estimates have shown, that one can expect a result like $\sigma_{natural} \gg \sigma_{artificial}$.

A short test example can demonstrate this: the distribution of the PageRanks of sites linking to the homepage of Steven Hawking[3] analyzed based on the distribution function above have a width of $\sigma^2 = 1.1$, while the sites belonging to a typical spam cluster have a PageRank distribution with $\sigma^2 = 0.5...0.7$[4]. The parameter σ can be used for separating between these two types of link clusters.

Further methods for detecting spam using link information can be found in Becchetti et al. (2006) and Gyöngyi et al. (2006). Saito et al. (2007) have compared different graph algorithms for spam detection in a large-scale study.

Clustering and more
Clustering algorithms based on link analysis will be described in Chapter 3. More graph-theoretic Web algorithms can be found in Deo and Gupta (2001a).

2.5 Data Structures

This section compares different data structures for storing the Web graph and analyzes their applicability on the HITS algorithm. As discussed in Section 1.3 and Subsection 2.3.3, the selection of data structures used to store Web content has a great impact on the possibilities of processing the data. In the context of the HITS algorithm, URL names, links with other pages in the base set, hub and authority values have to be stored for each page involved.

When evaluating data structures for their suitability, one has to take into account the following properties:

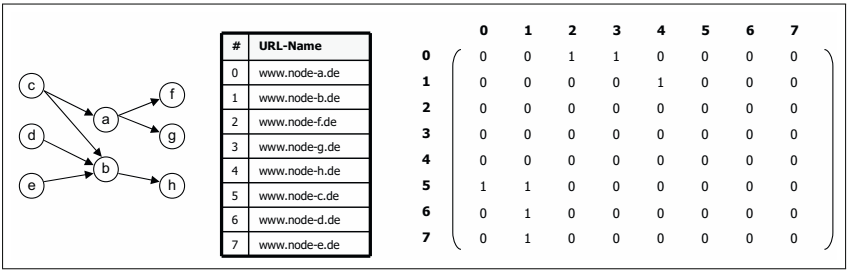
1. *Performance:* How fast can data be read or written, and how fast is data access?

2. *Scalability:* How do performance and manageability change with increasing amounts of data?

3. *Memory requirements:* How much memory is used by data structures, and how is its impact on performance and scalability?

4. *Compatibility to the ranking algorithm:* Is the data stored in a structure that is optimized for the algorithm used, or are additional data transformations necessary?

[3] `www.hawking.org.uk`
[4] The data of this example is based on the PageRank indicator of the GOOGLE toolbar (`toolbar.google.com/intl/de`).



#	URL-Name
0	www.node-a.de
1	www.node-b.de
2	www.node-f.de
3	www.node-g.de
4	www.node-h.de
5	www.node-c.de
6	www.node-d.de
7	www.node-e.de

	0	1	2	3	4	5	6	7
0	0	0	1	1	0	0	0	0
1	0	0	0	0	1	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	1	1	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0
7	0	1	0	0	0	0	0	0

Figure 2.16: Sample link structure with URL table and adjacency matrix A

2.5.1 Feasible Data Structures

Based on the list of requirements, three different data structures are chosen to be regarded in detail. After describing the implementation of a straight-forward approach of the matrix notation (compare Subsection 2.4.2), two alternative data structures will be described.

Adjacency Matrix

The subgraph of the base set $G(V,E)$ is stored in a $|V| \times |V|$ adjacency matrix A. The implementation of the adjacency matrix approach will be described using an example.

Figure 2.16 shows a sample link structure consisting of a subgraph of eight vertices, named with the letters `a` to `h`. The vertices correspond to the URLs `www.node-a.de` to `www.node-h.de`. The table in the center of the figure shows an assignment of matrix column and row numbers to the URL name.

The adjacency matrix A in Figure 2.16 consists of a row i and a column j for each page of the base set. Row i contains information of the outlinks of page `a`, i.e. the entry "1" in each column j for a page of the URL list page `a` is pointing to. An algorithm is able to perform calculations based on this matrix without accessing the base-set data.

For the calculation of an iteration of the HITS algorithm, only two additional array structures are needed to store hub and authority values besides the adjacency matrix. The authority values x_i of iteration k are calculated out of the adjacency matrix and the hub values y_j of iteration $(k-1)$ using the $\mathcal{I}$ operation to

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}_k = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^T \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix}_{k-1},$$

and the hub values using the $\mathscr{P}$ operation analogously to

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix}_k = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}_{k-1}.$$

Each update operation only consists of one matrix multiplication. In this case, the data structure is adapted to the one of the original algorithm. Two problems arise using the matrix data structure. The sizes of the base set and thus of the adjacency matrix are not known a priori, so that the algorithm cannot allocate the memory in advance. In practice, link densities of base sets are typically lower than 1 %. The consequence is a sparse adjacency matrix with many zero entries that lead to more than 99 % of operations without contribution to the result.

If one assumes a sample base set with 5'000 vertices and a link density of 0.5 %, resulting in an adjacency matrix of $5'000 \cdot 5'000 = 25'000'000$ entries, then $25'000'000 \cdot 0.995 = 24'875'000$ entries are zero entries. After 20 iterations of the algorithm, almost one billion irrelevant operations are performed ($2 \cdot 20 \cdot 24'875'000 = 995'000'000$).

A solution for both problems can be achieved by storing only those matrix entries that are relevant for the calculation.

Transformation into an Array Structure

In the following, a data structure for storing only relevant values of the adjacency matrix will be introduced. Two array structures are used to store inlink and outlink data. Even though both arrays contain the same information, the redundancy of using two arrays will be accepted for the sake of an easy access to both information types.

The array structures are filled as follows. The adjacency matrix is processed column by column from the left to the right side in order to extract inlink information. In the example, the vertex with number $i = 1$ has the vertices with numbers

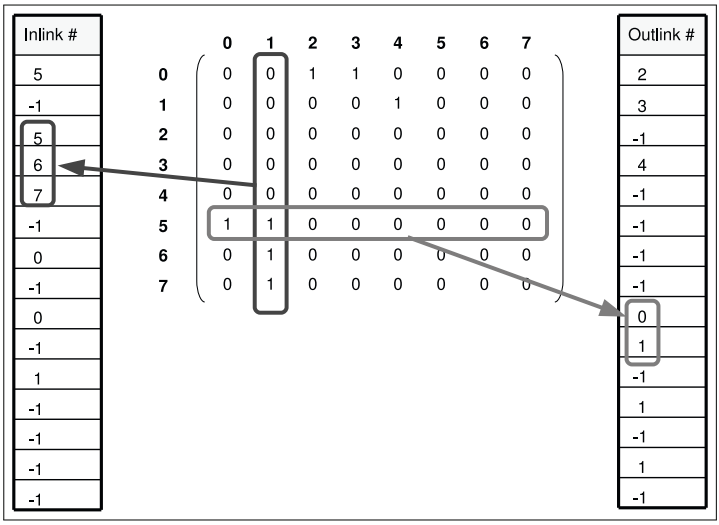


Figure 2.17: Transformation of adjacency matrix into two arrays

$j = 5$, $j = 6$ and $j = 7$ as inlink, thus the inlink array stores these numbers as shown in Figure 2.17. A separator value "-1" occurs in the array after the entries of each column's vertices. For creating the outlink array, the rows of the matrix are processed analogously.

Whereas the adjacency matrix of the example needs to store $8 \cdot 8 = 64$ entries, both arrays with 15 entries each (including separators) together consume only 30 entries, thus less than half of the memory consumption. In more complex link topologies the difference increases even more because the proportion of separators compared with the total of entries decreases.

For the HITS algorithm, each array only has to be processed once per iteration. During each iteration, the array only needs to be sequentially passed through, without any additional transformations or jumps. In order to calculate the hub value y_0 for vertex 0, the outlink array is read up to the first separator value. The resulting values are the outlinks $\{2,3\}$. Using these values in the formula of the $\mathscr{I}$ operation, the hub value calculates to $y_0 = x_2 + x_3$. The next value in the array is 4, which leads to the calculation $y_1 = x_4$. The complete set of calculations performed for each iteration resulting from the inlink and outlink array is shown in Table 2.1.

Even though the array structure significantly reduces the number of calculations, there are two remaining disadvantages. The array structure needs memory

#	Authority Value ($\mathscr{I}$ operation)
0	$x_0 = y_5$
1	$x_1 = y_5 + y_6 + y_7$
2	$x_2 = y_0$
3	$x_3 = y_0$
4	$x_4 = y_1$
5	$x_5 = 0$
6	$x_6 = 0$
7	$x_7 = 0$

#	Hub Value ($\mathscr{O}$ operation)
0	$y_0 = x_2 + x_3$
1	$y_1 = x_4$
2	$y_2 = 0$
3	$y_3 = 0$
4	$y_4 = 0$
5	$y_5 = x_0 + x_1$
6	$y_6 = x_1$
7	$y_7 = x_1$

Table 2.1: Calculation of authority and hub values based on the array information

additional to the adjacency matrix, and the transformation process needs a lot of computation time without any information gain.

List Structure

In order to overcome the problems of the array structure, a data structure without using an array for the adjacency matrix is presented in the following.

Two possibilities are considered to store the base-set data. First, each vertex of the base set can be stored together with their inlinks and outlinks. Second, the set of inlinks and the set of outlinks can be stored as two separate objects.

The first alternative results in a large number of lists in the memory (one per page) and thus in a worse performance. A better performance will be expected from the second alternative as only two instances (one for inlinks and one for outlinks) have to be accessed.

The array structure of the adjacency matrix allows for reading links from both directions, i.e. it is as easy to directly read the inlinks of any page as to read its outlinks. Modeling the data structures without an array still leads to maintaining two different structures for inlinks and outlinks to ensure a quick access, even though both structures contain the same information.

Storing the inlink and outlink information in an array using the above described line-by-line and column-by-column approach is not possible without an adjacency matrix array. Processing one link after the other leads to a situation that at least the starting vertices or the ending vertices of the links are not sorted in an ascending order. That means for the storage in an array, that the storage position, e.g. for an outlink, cannot be known in advance if there are outlinks not yet processed to be stored prior in the array. The problem of the unknown position can be solved by

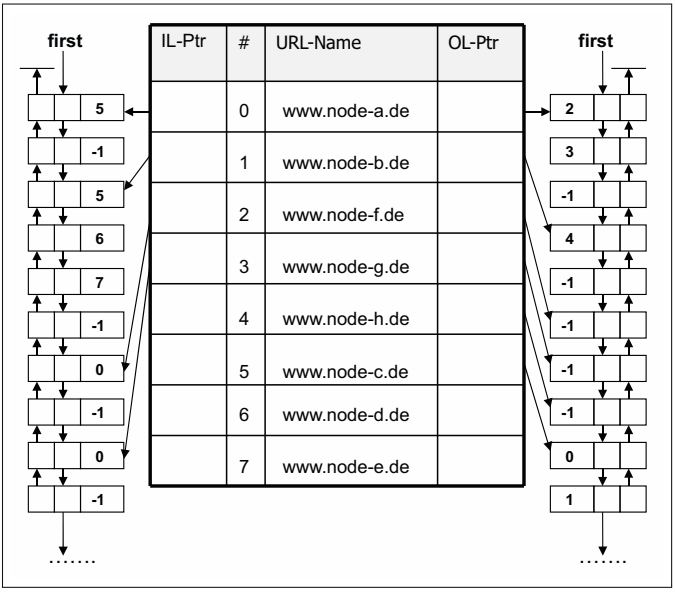


Figure 2.18: Graph modeling using doubly-linked lists

either leaving vacant spaces in the array to be filled later or by not leaving spaces and later – if necessary – perform insert operations on the array. The first solution destroys the memory advantages and the second solution the performance gains of the array structure.

Both problems can be avoided by using doubly-linked lists, because new elements can be inserted at arbitrary locations in the list. After creating an inlink and an outlink list, two pointers to elements of each list have to be stored for each URL like shown in Figure 2.18. The navigation through the lists can be performed analogously to the one used for the above described array structures. Thus, the adaption to the HITS algorithm is similar to the array from the transformed adjacency matrix. It needs less memory, because the lists' sizes only exhibit a linear growth with the base-set size. The lists can be filled without the time-consuming transformation process. The navigation through the lists is slower than through arrays. A more profound analysis of performance, memory needs and scalability is subject of the next subsection.

	Array Structure	List Structure
Base Set	one-dimensional string array	one-dimensional string array
Connection	pointers to array positions	pointers to list elements
Inlinks	one-dimensional integer array	doubly-linked list
Outlinks	one-dimensional integer array	doubly-linked list
Adjacency Matrix	two-dimensional Boolean array	-

Table 2.2: Evaluated data structures

2.5.2 Evaluation

Among the data structures presented in Subsection 2.5.1, the first one – the adjacency matrix – will be excluded from further consideration because of its scalability problems. The remaining two structures – the adjacency matrix transformed into an array structure and the list structure – will be evaluated based on their memory demand, their performance and their scalability. For this purpose, both data structures were implemented in a program calculating the hub and authority values with the HITS algorithm. Table 2.2 gives an overview of the main elements of both data structures.

Resources

The data structures and algorithms were implemented in C++ under MICROSOFT VISUAL STUDIO .NET 2003. For the measurements, a personal computer with an ATHLON XP 1800+ CPU and 256 MB RAM connected with a 266 MHz bus was used. WINDOWS XP was used as operating system. During the tests, only necessary processes of the operating system were active. Test times are determined using the `clock`-operation of the system clock in milliseconds (ms). Values lower than 10 ms cannot be captured and will be replaced in the tables by "< 10". The memory demand is determined using the task manager of WINDOWS XP and recorded in kilobyte (kB).

Experimental Data

For the following measurements, eight data sets of different sizes and link densities were created using arbitrary search queries and two different search engines. The

Set	Query	Search Engine	URLs in Root Set	URLs in Base Set	In-links	Out-links	Link Density
A	"Sport Basketball"	Alltheweb	25	422	93	364	0.257 %
B	"Rentenreform"	Alltheweb	500	1866	482	1791	0.065 %
C	"Riesterrente"	Alltheweb	500	2043	605	5880	0.155 %
D	"Suchmaschine"	Google	100	3811	2867	1359	0.029 %
E	"Suchmaschinen"	Google	100	3843	1388	3329	0.032 %
F	"Kleinberg Hits"	Google	250	4287	481	4134	0.025 %
G	"Lastminute"	Alltheweb	150	7656	6361	2035	0.014 %
H	"Search Engines"	Alltheweb	100	9065	4745	5943	0.013 %

Table 2.3: Test data sets

content of the query and the search engine used only have an inferior impact on the time measurement results.

Table 2.3 lists the examined data sets A to H. Based on root sets of 25 – 500 URLs, base sets of between 422 and 9'065 URLs are generated. Their link density

$$\frac{|E|}{|V| \cdot (|V|-1)}$$

is calculated as the number of edges $|E|$ of the base set graph $G(V,E)$ divided by the maximum possible number of existing links $|V| \cdot (|V|-1)$. The link density varies from 0.013 % to 0.257 %. The base sets were selected in a way that the relation from inlinks and outlinks varies, because the storage methods for both link types differ.

Memory Demand

The memory demand of both data structures depends on the preset maximum size (`MAX`) of URL sets, because the program reserves system resources based on this value. For a maximum size of `MAX`=10'000 URLs, the adjacency matrix needs an array of 10'000 fields in order to store the URLs and a matrix of 10'000·10'000 entries to store the link information. The list-based software starts with an array for the URLs and dynamically creates lists for the link structures during run-time.

Table 2.4 shows the observed memory demand of both data structures for different maximum sizes of URL sets during the experiments conducted. All values contain a fix minimum memory demand of approximately 7'500 kB for the user

URL Limit (MAX)	1'000	5'000	10'000	100'000
Array Structure	∼9'160 kB	∼57'000 kB	∼205'000 kB	·
Linked List	∼7'800 kB	∼8'900 kB	∼10'500 kB	∼28'500 kB

Table 2.4: Observed memory demand of different maximum base-set sizes

interface and methods. The memory demand for the array structure of 100'000 URLs could not be experimentally determined due to memory restrictions of the hardware used. This memory demand is estimated to be ≥20'000 kB.

Whereas the array structure's memory exhibits quadratic growth with the value MAX, the linked list's growth is linear, both adjusted by the fix minimum memory demand of 7'500 kB. The linked list's memory demand additionally depends on the actual link structure and is hardly predictable in advance.

Performance and Scalability

For time-measurement purposes, several breakpoints are set into the program flow, in a way that other operations like read or write to harddisk or Web access are excluded. The starting and ending points of a measurement are always placed directly before and after the corresponding operation, respectively.

The breakpoints are explained in the chronological order of the program flow. The first breakpoint is placed in the load routine of the root set. It measures the time the program needs to store links into memory using the `append` method. Due to the low quantity of URLs processed, the durations are lower than the minimum measurable time of ten milliseconds and thus cannot be considered.

The next two breakpoints are placed in the load routine, where inlink and outlinks are stored in the respective data structure. Each set was processed three times, and the mean values of processing times between the start and the end of the `append` method were calculated. Table 2.5 shows the processing times of loading the inlink and outlink information of each test data set in the array structure as well as in the linked list.

Larger data sets need more time to append all links (inlinks and outlinks) than smaller sets. The processing time exhibit an approximately quadratic growth with additional URLs. A reason for this observation is that each newly found URL has to be verified whether it already exists in the base set. Moreover, a multiple insertion of the same link has to be avoided. For this reason, the set of already captured links has to be scanned for the existence of the new link.

Set	Store Inlinks (append) (ms)		Store Outlinks (append) (ms)	
	Array Structure	Linked List	Array Structure	Linked List
A	44	48	70	100
B	791	801	1'533	1'762
C	1'072	1'154	3'235	3'853
D	7'761	7'962	1'062	1'202
E	4'946	4'932	3'996	4'577
F	2'003	2'063	9'229	10'305
G	35'632	36'924	2'304	2'364
H	36'263	38'495	5'547	5'679

Table 2.5: Duration of load routines

A significant difference between the results of inlinks and outlinks can be noticed. This difference results from the program sequence where the outlinks are always added to the set earlier than the inlinks. The verification for existence of a URL has to be performed at the time inlinks are added. The probability of a URL already being in the set increases with the number of links between sites. The time the verification needs depends on how quick the URL is found, what again depends on how early the URL is found in the array or list.

Regarding only the outlinks, the array structure is always faster than the linked list. Its speed advantage decreases with a growing number of links to add. This behavior can also be explained with the time consumption of the existence check.

The next measurements are performed during the navigation through the data structures. Reading the complete data set is for example necessary during the presentation of the link structure on screen, to store it to harddisk or to perform a search on the set. The breakpoints are placed into three memory routines, and their times are added up in order to gain one value for the complete pass of the navigation.

Table 2.6 reveals a clear advantage of the list structure over the array structure. The list structure needs considerably less operations for this task because it does not have to run through all fields of the matrix for checking the contents. It is sufficient for the list-based software to only sequentially process list entries for each root URL and jump to the next URL. Because of the above described limits of the system clock, the duration needed for one additional URL cannot be determined. The number of operations allows an estimation of the behavior of both algorithms.

Set	Navigation through Inlinks (ms)		Navigation through Outlinks (ms)	
	Array Structure	Linked List	Array Structure	Linked List
A	<10	<10	<10	<10
B	141	<10	119	<10
C	160	<10	152	29
D	80	<10	50	20
E	70	11	61	21
F	231	10	150	21
G	361	16	151	26
H	291	15	132	40

Table 2.6: Duration of navigation

Using the array structure, the number of operations exhibits a quadratic growth with the number of URLs. The number of operations on the list structure depends on the number of links stored. It is at least as high as the number of root links. Only in the worst case, that is a complete linkage of all pages, the same amount of operations as for the array structure is needed. Thus, the processing time of the list based software grows by a smaller or the same amount compared to the array based software.

The last breakpoints are set during the ranking algorithm. They are divided into the duration of initialization (transformation) of the data, the ranking algorithm itself, and the sorting of values and output in form of an array. The measurements are performed with fifteen iterations each. For the sake of clarity, Table 2.7 shows the totals for separately measured times of hubs and authorities.

For the list structure, the data transformation is not necessary, and, consequently, it does not consume any time. The iterations of hub and authority calculation using the array structure only takes about half of the time the list structure uses. The time for sorting the values and creating an output vector is about the same for both structures. Summing up all values, the time the array structure spends for the data transformation exceeds the one it saves during the iterations by a multiple.

2.5.3 Selection

In order to select a suitable data structure to store Web graphs for ranking algorithms, i.e. the HITS algorithm, all measurements are regarded together. From a

Set	Transformation (ms)		Iterations (ms)		Sort and Output Vector (ms)	
	Array Structure	Linked List	Array Structure	Linked List	Array Structure	Linked List
A	40	0	18	21	120	120
B	1'091	0	31	78	2'383	2'407
C	1'302	0	61	139	2'502	3'042
D	5'228	0	52	141	8'441	8'562
E	5'328	0	61	151	8'922	9'076
F	7'630	0	59	150	9'714	9'812
G	34'322	0	103	261	34'677	35'211
H	48'957	0	110	329	45'201	45'409

Table 2.7: Duration of ranking algorithm

Array Structure	List Structure
↑ proximity to mathematical notation ↑ simplicity of implementation	↓ implementational complexity
↓ high memory demand ↓ inefficient on low link density	↑ low memory demand
↓ quadratic memory growth	↑ linear memory growth
↓ quadratic time consumption	↑ speed

Table 2.8: Advantages (↑) and disadvantages (↓) of evaluated data structures

practical point of view, already the smaller memory demand of the list structure makes it preferable to the array structure. It also scales better in terms of memory and speed. The higher speed of the array structure during the ranking calculation does not carry that much importance because this calculation only takes about ten percent of the complete processing time.

The higher implementation effort of the list structure pays off in better runtime performance regarding time and memory. An overview of advantages and disadvantages of both data structures is outlined in Table 2.8.

2.6 HITS in Practice

This section describes practical impacts of the HITS algorithm. For this purpose, an own implementation of the algorithm was programmed and tested. It is used to evaluate both, data structures and result quality. The program calculates hubs and authority values of the results returned for a given search query.

2.6.1 Implementation of the Algorithm

The HITS algorithm is implemented in the programming language C++. The goal of the software is to retrieve hub and authority websites fitting to a given user query.

The program's process flow is triggered by a query entered by the user in a field of the user interface. The information flow of the process is explained using Figure 2.19. The query is sent to a public available search engine that has prepared data from the Web at its disposal. Here, the search engines GOOGLE and ALLTHEWEB are used. The URLs of the top n pages are parsed from the search-engine result-page and are stored in memory. These URLs build the root set. In order to expand the root set to a base set, those pages linking to pages of the root set (inlinks) and those pages linked-to by pages of the root set (outlinks) are needed. Both types of links are usually not contained in the result page.

Inlinks and outlinks are gathered using two different methods. As the inlink information is not contained on the regarded website, an external source is needed. Search engines deliver URLs of those pages linking to a specified page when queried with the `link`-operator. During the program flow, these URLs are requested from the search engine, and the result page is again parsed. The inlink information is being stored in the chosen data structure. The outlink information is contained in the HTML code of the webpages belonging to the root set. These pages are downloaded directly from the Web. The outgoing links are parsed from the HTML code and stored to the chosen data structure.

As soon as all needed inlinks and outlinks are stored, the iterative hub and authority value calculation starts according to Algorithm 2.2. The results are sorted and then presented to the user on the internal result page.

Classes

The above described process flow is implemented into nine C++ classes. The classes can be assigned to five areas. Figure 2.20 shows the following class areas, together with their access possibilities that are marked with arrows:

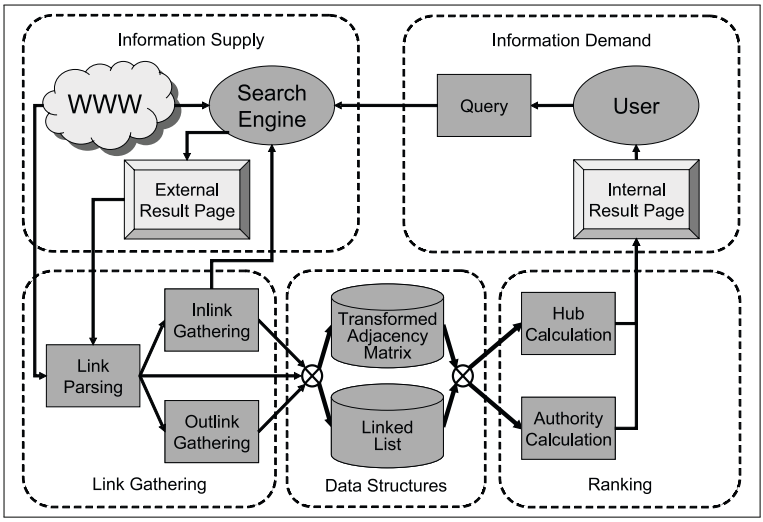


Figure 2.19: HITS architecture – information flow

1. *Main:* This area contains one class that serves as main class and as entry point of the program.

2. *User Interface:* Three classes perform the dialog with the user.

3. *Access Control:* One class provides Web access and harddisk storage operations.

4. *Data Structures:* Two classes provide the data structures.

5. *Ranking:* Two classes perform the ranking calculations.

At program start, the class `RankDataDlg` instantiates an object of the user interface which creates the application's main window. It is able to create objects of the classes `SetCrawlerDlg` and `RankDlg` that serve for the entry of settings and for the output of ranking results.

Both classes access the class `CrawlFunctions` in order to start the parsing of search-engine result-pages or webpages respectively and to change their settings. The class `CrawlFunctions` stores the links gained into memory using the data structures classes. The class `Memory` defines the data structure used and provides access functionality to the data, like inserting URLs into the root set. Depending

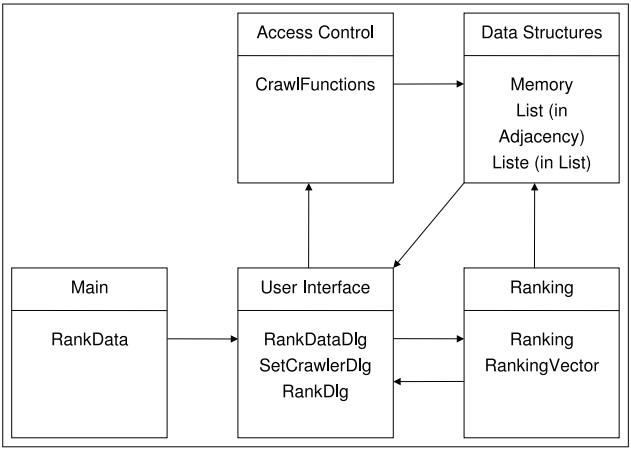


Figure 2.20: HITS Software architecture – class view

on the data structure chosen, the class `List` is used for the administration of the base set in form of either an adjacency matrix array or a linked list.

The class `Ranking` accesses the content of memory and lists in order to calculate hub and authority values. It is triggered by the class `RankDlg` of the user interface. An object of the class `RankingVector` is created that contains the results in sorted order.

Filling Data Structures

The data structure implementation will be described on the basis of the linked list, which has proved to be superior to the array structure (compare Subsection 2.5.3). However, both data structures were implemented for the purpose of evaluation.

The class `Memory` provides storage capabilities for the root set. It consists of an array of memory elements, containing for each URL its name, a pointer to the first element of the list of outlinks and a pointer to the first element of the list of inlinks.

The following methods grant the access to the linked lists storing the focused subgraph. The `append` method inserts an URL in the main list if it is not yet part of the list. The `search` method returns the storage position where the name of a given URL can be found. Two methods can be used to append URLs as links to the lists. The `appendOutlink` method checks whether the URL already exists in the main list. If not, it is inserted in the main list. The URL is appended to the outlink list of the referencing URL. The referencing URL is inserted in the inlink

list of the added URL. The `appendInlink` method analogously adds the URL to the main list, if necessary, and updates the link lists.

The link lists are designed as doubly linked lists. The `list` class provides methods to append and to get values from the list. After the insertion of the URLs and their linkage information, the new URLs are tested for links among each other. Detected links are added to the corresponding lists.

Crawl Functions

The class `CrawlFunctions` fills the data structures with root URLs, inlink URLs and outlink URLs. It provides methods for

1. the access to search engines,
2. the parsing of search-engine result-pages,
3. the direct access to other webpages,
4. the parsing of link URLs from webpages, and
5. the input and output of data to harddisk.

The class `CrawlFunctions` creates objects of the type `Memory` and `Ranking`, and uses their methods to display the results on the user interface.

The methods `getAlltheweb` and `getGoogle` create an adapted HTTP-request string for the access to search engines. This string corresponds to the string a search engines HTML page sends to its server after the user has entered the query and pressed the search button. It contains the same information in the same format, and must be adapted to the chosen search-engine's semantics. Using this string, the method `getHtmlSource` returns the result page. If the search engine is used to deliver inlink information, a switch makes sure that the string contains the corresponding link command.

The methods `parseAlltheweb` or `parseGoogle` parses the result page for hyperlink information contained. These methods have to be adapted to the chosen search-engine's semantic, too. Navigational links and links created for advertisement purposes are excluded. Also secondary URLs, also referred to as "more results from the same page", are ignored in order to make sure that only top-ranked pages are considered.

Links from webpages belonging to root-set URLs can be easily gathered using the following two methods. The method `getHtmlSource` retrieves the HTML source of a webpage and the method `parsePage` searches it for outlinks in this source.

Three methods each for loading and writing URLs and data from and to harddisk are provided. They can store the results as well as the subgraph and retrieve them for further investigations.

Algorithm

The `Ranking` class stores hub and authority values in two arrays each. In order to avoid copy operations between the iterations, two arrays exist for each type of values that are alternately used for current and past ranking values.

Before the iterations start, ranking values are initialized to $x_i = 1$ and $y_i = 1$ ($\forall i = 1, \ldots, n$) using the method `init`. Authority and hub values for URLs from the main list are stored at the corresponding position number. Ranking values can be calculated without accessing the main list. The method `iterateRanking` performs the ranking calculations. It determines the hub value of entry i from the sum of those authority values contained in the outlink list of entry i (see Algorithm 2.2). The authority value of entry i is calculated from the sum of those hub values contained in the inlink list of entry i. After an iteration, both the authority and the hub vector are normalized to $\sum_{i=1}^{n} e_i = 1$ using the method `normalize`.

After the iterations are performed, the result values are connected with the corresponding URL names using the class `RankingVector`. It produces an array containing URLs and ranking values, which is sorted by the method `makeSort` using the bubble-sort algorithm in order to present the URLs in a sequence starting with the best result and ending with the worst result.

User Interface

Three windows constitute the user interface: a *main* window, a *settings* window and a *ranking* window. The main window contains a query entry mask and three output areas for root links, inlinks and outlinks. A screenshot of the main window containing sample data is shown in Figure 2.21. Some parameters like the number of links in the root set, the number of iteration steps, the maximum number of outlinks considered per domain, and the name of the search engine used can be set in the main window. A check box for using extended links can expand the root set to the URLs of the base set, so that an additional level of link distance can be used.

Additional parameters can be set in the settings window. The ranking window shows two output areas for the sorted output of URLs and their hub values. Figure 2.22 shows a sample output of this window.

2.6.2 Experimental Results and Evaluation

The implemented software is used to evaluate the practical relevance and the functionality of the HITS algorithm. One of the algorithm's strengths results from the possibility to answer broad queries by identifying authorities and hubs. Even if the first results do not contain good authorities for a query, the user may find some good authorities in the link list of a high-ranked hub. Showing the authority and hub values provides the user with additional information about the results' goodness in order to improve his judging potential.

Description of Page Content

The PageRank algorithm only uses linkage information of a Web subgraph to determine ranks. HITS expands this approach by using content information additionally to link information. The expansion of the root set by considering inlinks and outlinks set the basis for the following observation. The HITS algorithm is able to find pages with high hub values that did not appear in the original root set. These hub pages contribute valuable link lists relating to the search query. The retrieved hub pages sometimes outperform search engines in terms of quality, structure, and visualization of their links.

A query for the term "`search engine`" on ALLTHEWEB, for example, brought as result a few meta search engines, but did not show authorities like GOOGLE or ALTAVISTA. Using link based information of the subgraph, HITS included these search engines in the base set and ranked them with high authority values.

Topic Drift

Within the evaluation of results, some weaknesses of HITS were observed. One problem is caused from different prevailing topics of root set and base set. This phenomenon is known as *topic drift* (Bharat and Henzinger, 1998) and may occur if pages added to the root set deal with differing topics.

A query for German politics ("`deutsche politik`") illustrates these observations. The root set of one hundred pages contains several organizations like research institutes, federal bank, and foreign office. These results are followed by 21 links to German embassies throughout the world. The base set did only contain three regional webpages of political parties. Applying the HITS algorithm on this data, the top-twenty authority-values are concentrated on German embassies. The focus on embassies overlays other topics because of their strong presence in the root set in connection with a strong linkage among each other and links from different hubs. The hub ranking does not show one single page of the root set among the ranks one to five. Among the top-ten hub-pages, there is only one page of the

original result page. On the second place of hub values, there is a page containing hundreds of links to ministries and political parties.

Sometimes, the authorities of the original search query are replaced by related authorities with a higher global impact and a higher link density in the subgraph. This case can be characterized as a "zoom-out" of the original topics onto a broader level. This phenomenon is illustrated using a search query for the name of a young and hardly known rock-band. The GOOGLE search delivered the band's official website and a few CD reviews and CD offers. The expansion of the root set has added some music clubs to the base set. These clubs appeared on the first authority ranks and contained a few other, more popular bands. Among the top hub-pages were link lists that point to bands of the same genre. Thus, the algorithm expanded the horizon to geographically and musically related bands.

However, a generalization or complete drift of the context often is not what the user wants and may lead to unsatisfying results.

Spamming

If webpages in the result set of a query use techniques to artificially manipulate their ranking in search engines (spam), this can negatively affect the HITS result quality. Spammers create for example different websites and provide these sites with links among each other for the only purpose of gaining high ranking values from link-based search engines. These websites do not necessarily contain relevant information concerning the keywords on their pages.

In the HITS results, such pages are able to displace "real" authorities and hubs from higher ranks. After the initial search query has generated the root set, there is not any further content evaluation performed by the HITS algorithm. Thus, the algorithm is not able to discriminate between valuable information and spam. The denser the internal link structure of the spam pages compared to the remaining base set is, the higher is its impact on the final result.

A possible solution for this problem is to limit the number of links on the same domain when creating the base set. In this way, the links' influence on the ranking can be reduced.

Link Concentration on same Domains

An analogical problem may occur if pages of the root set contain many navigational links to pages of another domain. Such a link structure brings high authority values to the referenced pages which again lead to a high hub value of the referencing page. The probability of links set only for navigational reasons to belong to the same topic is low. This is especially the case if the target domain does not contain any relevant information, like it is in the case of advertisements.

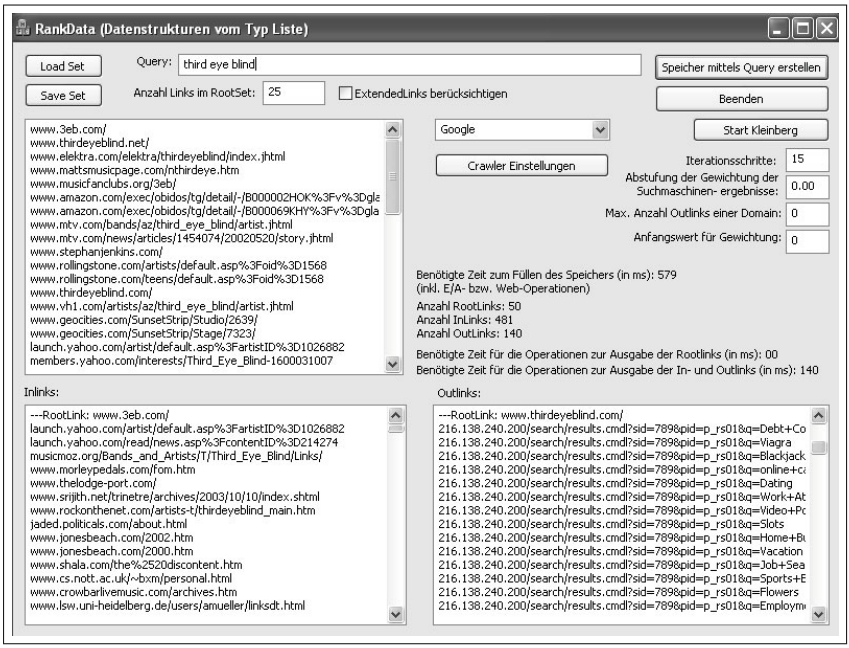
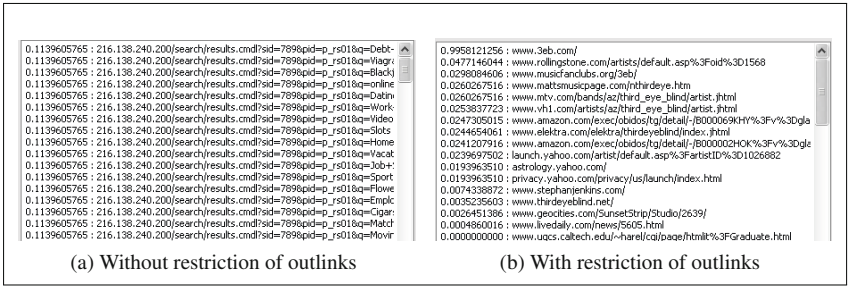


Figure 2.21: Main window showing outlink concentration on same domain



(a) Without restriction of outlinks (b) With restriction of outlinks

Figure 2.22: Authority values before and after restriction of same domain's outlinks

Again, a solution is to set a limit for the number of outlinks to the same domain for each entry in the base set. The number of inlinks coming from the same domain is limited, too. Implementing these limits as an optional parameter in the HITS software improves the ranking quality as shown in the following example.

A sample search without the link-limiting parameter set comes to the results in Figure 2.21. The lower right box of the main window shows a remarkable number of outlinks pointing to the same domain `216.138.240.200`. The outlinks contain strings that are not obviously related to the query. A closer look revealed that these links belong to a search engine and refer to categories of their site. Although they do not contribute any value to the result, the search engine's pages occupy the highest authority ranks (see Figure 2.22a) because of their link structure. Running the algorithm with a restriction to a maximum number of links per domain leads to the authority values of Figure 2.22b, which provides a more realistic image of the authorities. The results with authority values greater than zero do not contain any obvious spam.

3 Result Clustering

This chapter describes various methods for structuring search results in the World Wide Web. The emphasis lies on grouping *(clustering)* of similar search results and their clear illustration.

The common purpose of clustering methods is the division of data into groups of similar objects. Grouping of Web search results according to their topic will help to achieve a better overview over the available information. In the case of search results belonging to different topics, clustering methods help identifying different topic groups and displaying them in a clearly arranged manner.

The following example illustrates a typical case for clustering methods applied on Web search results. A query for the term "Burlington" returns results from a wide range of results from different topics associated with the name "Burlington". Among the results are a few tens of towns worldwide bearing this name and websites related to this town as well as a textile manufacturer registered under this same name. Persons with the surname Burlington are following on lower ranks and hence are hard to find. Here, the limits of a classic PageRank oriented search engine become visible. It can only determine one overall relevancy value for each result. In this case, the PageRank calculation is not able to differentiate between webpages related to the towns and those related to the persons.

Clustering methods can help to overcome these weaknesses. They are able to find similarities among these webpages belonging e.g. to the same town or the same person, respectively. This information can be used for grouping the results according to the similarities found, and it improves usability.

After presenting some common clustering methods in Section 3.1, the requirements for adapting these methods to Web search are analyzed in Section 3.2. Section 3.3 presents existing text-based approaches for clustering of Web search results. In Section 3.4, new clustering algorithms based on the link information in the Web graph are introduced, implemented and evaluated. Several combined approaches of text-based and link-based methods are arranged and tested using a common architecture and are evaluated in Section 3.5.

3.1 Clustering Methods

A variety of different clustering methods for multiple purposes has been published. Berkhin (2006) provides a comprehensive overview about existing clustering techniques.

Traditionally, one distinguishes between *hierarchical* and *partitioning* clustering methods. Hierarchical methods either start with single points (data items) and gradually assemble them into clusters *(agglomerative* or *bottom-up approach)* or start with one cluster containing all points and gradually disassemble them into clusters *(binary divisive* or *top-down approach)*. Different approaches are used by *partitioning* methods. They aim to directly identify clusters. For this reason, they either iteratively relocate points between subsets *(partitioning relocation clustering)* or directly identify areas heavily populated with data *(density-based clustering)*.

The hierarchical clustering uses the following procedure. In the agglomerative case, at the beginning each point builds a singleton cluster. In each iteration, the two most appropriate clusters are chosen and merged together. The process continues until either a stopping criterion is achieved or all points belong to only one cluster. In the divisive case, all points build one common cluster at start. In each iteration, one cluster is split into two new clusters following a decision criterion. This process continues until either a stopping criterion is fulfilled or each cluster consists only of one point.

In order to decide about which clusters to merge or to split, distances or similarities between subsets have to be calculated. The distances between subsets of points can be derived from the distances of the single points in different ways. If $d(x,y)$ is a distance measure between the points x and y, derived distances between two clusters C_1 and C_2 can be calculated with different linkage methods, here determined by a mutable operator Op:

$$d(C_1,C_2) = Op\{d(x,y)|x \in C_1, y \in C_2\}.$$

Depending on the operation Op different concepts of closeness and connectivity can be modeled. This includes e.g. the single link case ($Op = min$), where the distance of two clusters is determined by those two points belonging to a different cluster each, that have the lowest distance. In the average link case ($Op = avg$), the average of the distances over all pairs of points from different clusters is calculated. The complete link method ($Op = max$) takes the maximum distance between two points of the clusters as the cluster distance.

Hierarchical methods exhibit a high flexibility regarding the level of granularity because the clustering process can be stopped at any depth of the resulting tree. They can easily handle any form of similarity or distance, and, consequently, they are applicable to any attribute types. A problem is to establish an appropriate termination criterion. A hierarchical algorithm adapted for categorical data is introduced by Guha et al. (2000) and will be deployed in Subsection 3.4.

Partitioning methods (also and more precisely known as *partitioning relocation clustering*) directly divide data into several subsets. They use heuristics in the form of iterative optimization because checking all possible subset combinations is in general not feasible in reasonable time. Different relocation schemes iteratively reassign points between a preset number of clusters. A good representation for the underlying data can lead to clusters that allow clear interpretations.

Probabilistic methods fall in the category of partitioning methods. They try to characterize and identify the statistical distributions the data may be constructed of. The functionality of probabilistic clustering will be analyzed in more detail in Subsection 3.5.4.

The partitioning methods k-medoids and k-means build a preset number k of clusters by minimizing the distance to a representative. One of the clusters' points represents the cluster in the k-medoids methods, whereas k-means uses the weighted average of the cluster's points as representative for the cluster. The weighted average can also be interpreted as the centroid of the cluster. If minimizing the sum of squares of errors between the points is the objective function, the k-means methods can be regarded as a special case of probabilistic clustering.

Representing the data points as vertices of a graph and assigning distances as edge weights, a clique-partitioning approach can be used for dividing the graph into clusters. Dorndorf and Pesch (1994) have implemented this approach in an ejection chain algorithm that is deployed on the Web in Subsection 3.5.4.

For the clustering of documents, *binary divisive partitioning* methods are particularly suitable (Berry and Browne, 1999; Steinbach et al., 2000). Cutting et al. (1992) have developed procedures for the clustering of larger document sets and have tested them for news collections. The documents are represented in vectorial shape, weighted with the tf-idf[1]-scheme and set into relation to each other by using the cosine measure. In the *Vector Space Model* (Salton, 1989) documents are represented by vectors in the Euclidean space whereas each term corresponds to a vector dimension. Manning et al. (2008) explain the usage of basic clustering methods on the Web. A survey containing clustering methods for the Web is presented by Berry and Castellanos (2008).

[1] term frequency/inverse document frequency (Salton, 1989)

3.2 Web Clustering

Applying clustering methods on Web search poses several challenges. In order to establish efficient clustering mechanisms working on large document sets, decisions about the sort and the scope of the base data used have to be made. Public available search engines that offer clustering facilities run their algorithm on a subset of Web documents that is small enough for running cluster algorithms in a reasonable time. If they do not maintain an own searchable index, they typically make use of the results returned by other search engines in the same way meta-search engines do. In this case, they use the few text lines of information contained in the short descriptions (also called *snippets* or *teasers*) returned on the search-engine result-page.

Base data
Clustering of webpages can be performed on different types of base data. Besides the textual information of a webpage and the snippets generated by search engines, the link information of the Web graph can be used as base data for the clustering process. The link information can be – like the snippets – gathered from search engines that offer the output of incoming links (inlinks) for a webpage in their index. Under the assumption that webpages connected with hyperlinks are with a certain probability also topically linked with each other, this information can be used for clustering purposes, too.

In comparison with the textual information of a webpage, the exploitation of the hyperlinks is independent of the webpage language. A link between two pages dealing with the same topic but that are written in different languages can contribute to the creation of clusters of good quality. The inclusion of link information may also help to distinguish ambiguous terms like in the “Burlington” example.

Requirements
Zamir and Etzioni (1998) have defined requirements for clustering methods regarding Web search. Many conventional document-clustering algorithms are carried out offline and on complete documents. In the Web, the number of documents to be clustered is very extensive, and the documents quickly change. Assuming that clustering is carried out on a computer that is independent of the search tool and that gets search results as an input, the following characteristics are demanded:

1. *Relevance:* Documents that are relevant to the search query should be separated from irrelevant documents.
2. *Clear cluster descriptions:* The user should be able to recognize at first sight the content of the cluster.

3. *Overlapping:* If documents cover different ranges of topics, an assignment to several clusters should be possible.

4. *Snippet-Tolerance:* Based on snippets given by search engines high-quality clusters should be yielded.

5. *Speed:* Clusters should be delivered within a few seconds as the Web user expects short reply-times.

6. *Incremental procedure:* Each snippet should be processed as soon as it was received from the Web in order to save time.

3.3 Text Based Approaches

Especially for the clustering of Web documents, a procedure that firstly extracts descriptions for each cluster and afterwards assigns documents to the according descriptions *(Cluster Description First)* was developed by Zamir and Etzioni (1999). As basis for the clustering, all phrases that are shared by at least two documents are determined first. Later, exactly those documents that contain the cluster's description as a phrase are assigned to the cluster. A suffix-tree is used as data structure for storing the phrases (Baeza-Yates and Ribeiro-Neto, 1999). A good running time can be achieved by not taking the whole number of documents as a basis for the clustering, but only the snippets yielded by a conventional search engine.

SHOC
The approach of *SHOC (Semantic Hierarchical Online Clustering;* Zhang and Dong, 2004) extracts phrases for the description of cluster contents, too. This approach determines the underlying concepts of a document set using the *Latent Semantic Indexing (LSI;* see Deerwester et al., 1990 and Subsection 3.5.3) for the production of clusters. Natural language offers various alternative possibilities of expression for the formulation of a certain *concept* (or *circumstance*). The aim of LSI is to identify the underlying concepts of the documents by using *Singular Value Decomposition (SVD;* compare Subsection 3.5.3).

LINGO
The *Label Induction Grouping Algorithm* (LINGO; Osinski et al., 2004) uses a singular value decomposition approach, too. Different to SHOC where SVD is used for the identification of cluster contents, LINGO uses SVD for the determination of suitable concepts and cluster names. The procedure of LINGO can be divided into four phases:

1. *Snippet Acquisition:* During this phase, a search query is sent to a search engine available on the Web and the received snippets are stored.

2. *Preprocessing:* During this phase, first HTML-Tags are being filtered (text filtering) and special characters (e.g. "$", "%", "#") are deleted, except for punctuation marks (e.g. ".", "?", "!"). Punctuation marks will be used later on within the scope of phrase identification. Then, the language of the snippets is determined (*language identification)* based on *stop words* by using the *small word technique* (Grefenstette, 1995). If there is a stemmer (Porter, 1980) available for the according language, this stemmer will be used in order to treat derived forms of a word as a single term *(stemming)*.

3. *Feature Selection:* After identifying words that do not contribute much to the content *(stop words marking),* the third phase called *feature selection* begins. In this phase, as complete as possible coherent phrases are identified (*Phrase Discovery Algorithm;* Zhang and Dong, 2004). This helps to avoid an extraction of incomplete fragments of phrases. Afterwards, possible cluster descriptions are generated from the texts. For this purpose, a term-document matrix is built first by using the tf-idf scheme (compare Subsection 3.5.3). On this matrix, a singular value decomposition is carried out in order to gain concepts of the documents *(abstract concept discovery)*. These abstract concepts are compared to earlier extracted phrases as well as to terms using the cosine measure *(phrase matching).* As a result, those phrases that best represent the respective concept are gained. Cluster descriptions that are too similar to each other are eliminated *(candidate label pruning)*. This is the case if the cosine of the angles between two cluster description vectors is too small.

4. *Clustering:* In the fourth phase the snippets are assigned to cluster descriptions. For this reason, the vector space model (VSM) is used again. Instead of a search query, the snippets are matched with the cluster descriptions *(cluster content discovery)*. Finally, the clusters are sorted by using a cluster score that puts relatively well described and extensive groups at the top *(final cluster formation)*.

VIVISIMO and KARTOO

Examples for publicly available search engines that provide clustering capabilities are VIVISIMO and KARTOO.

VIVISIMO works as a meta-search engine using the data of other search engines and showing the grouped results as a hierarchical tree. VIVISIMO deploys

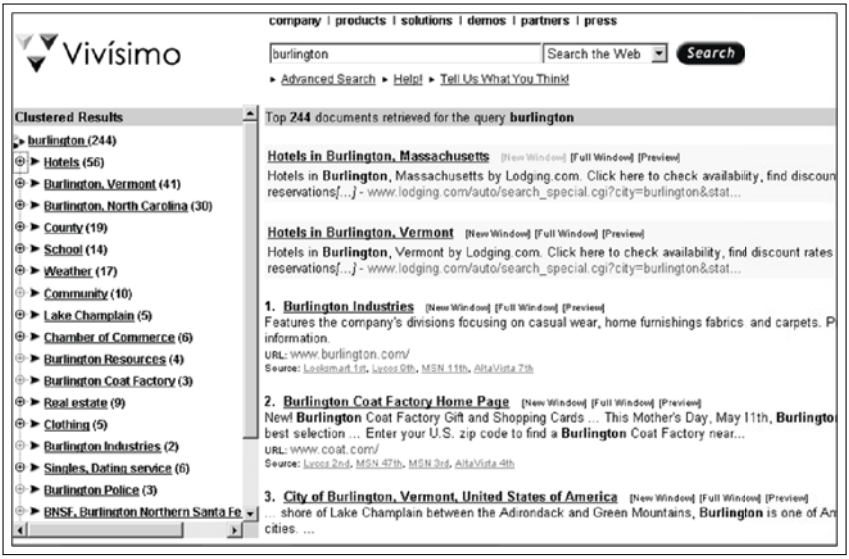


Figure 3.1: Sample result window of `vivisimo.com`

a cluster-description-first approach and yields quite a good quality of cluster descriptions. Applying this procedure, cluster descriptions are generated before the identification of clusters. In a second step, the documents are assigned to these descriptions denominating clusters. Figure 3.1 shows sample output of VIVISIMO.

KARTOO also works as a meta-search engine. Here, the search results are graphically shown in a two-dimensional area. The groups are represented through the proximity of results to each other, through differently colored coverings as well as through connecting lines between the results (see Figure 3.2).

Already when taking a superficial view the quality of results does not reach the results of VIVISIMO. Neither the authors of KARTOO nor the authors of VIVISIMO give further information about the kind of procedures used.

Other Methods

Broder et al. (1997) introduce a clustering algorithm that is based on fingerprints that are generated from the textual content of a webpage. Clustering algorithms using both, text and link information are developed by Modha and Spangler (2000). Zamir and Etzioni (1998) evaluate different methods of Web clustering.

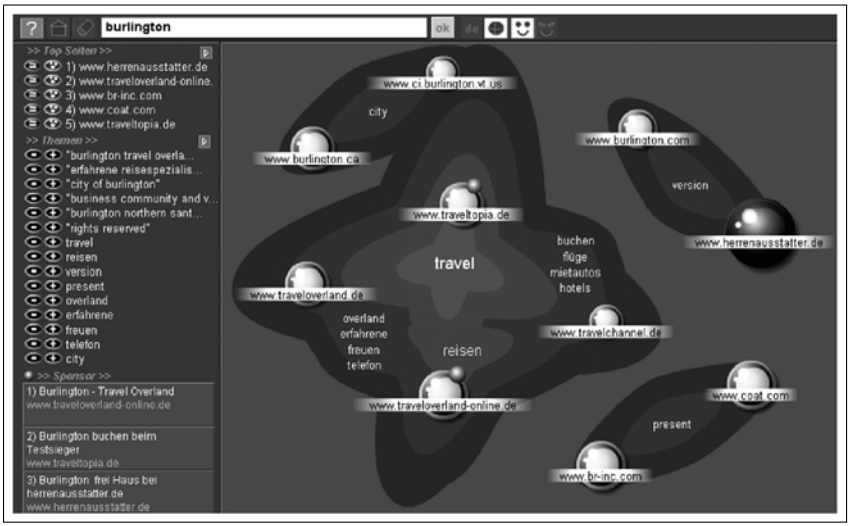


Figure 3.2: Sample result window of `kartoo.com`

3.4 Link Based Approaches

Besides the link analysis approaches shown in Section 2.4, the link structure of the Web graph can be used for clustering webpages. Assuming that two webpages connected with a hyperlink are more probably belonging to the same topic than two webpages not linked to each other, a measurement of the quality and quantity of links can serve as a distance measure for the clustering process. Webpages with a higher link density among each other can be joined to a cluster as shown in the example of Figure 3.3. The big circles group webpages with multiple links within the circle to a cluster, whereas the number of links between the big circles is relatively small.

Clustering based on the linkage graph among websites brings several advantages. As the link information does not contain any language specific properties, the clustering can be performed independently of the language the webpage content is written in and independently of the query language. This leads to another advantage. If the query contains ambiguous words, the link based clustering is able to keep apart the different meanings.

In this section, we introduce a new approach to purely link based clustering.

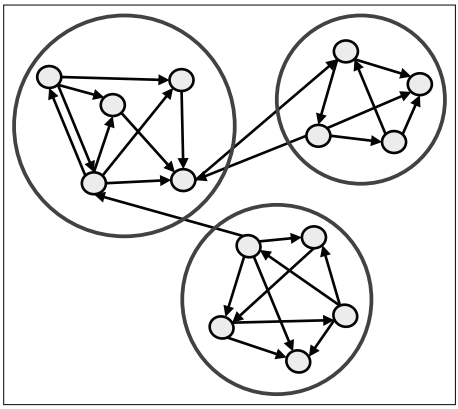


Figure 3.3: Webpage grouping by linkage

3.4.1 Robust Clustering Algorithm

An algorithm that has originally been created by Guha et al. (2000) for clustering of categorical data is particularly suited for hyperlinks because of his data representation: The algorithm ROCK (RObust Clustering using linKs) differentiates between *neighbors* and *links* of points. In the following, these links are called *ROCK-links* in order to avoid confusion with hyperlinks. Two points x and y are defined as neighbors if the value of a similarity function $sim : X \times Y \rightarrow \mathbb{R}^+$ exceeds a certain threshold $sim(x,y) \geq \theta$ $(0 \leq \theta < 1)$. Later, the comparison of the similarity value with a threshold will be replaced by different definitions adapted for the Web graph. Thus, the function *sim* will not be needed anymore. In any case, the number of common neighbors of two points x and y is denominated as ROCK-link $rocklink(x,y)$.

The algorithm performs hierarchical agglomerative clustering and produces clusters with a high degree of connectivity. It stops the agglomeration if a specified number $k \in \mathbb{N}$ of clusters is constructed. The ROCK algorithm maximizes the following objective function

$$\begin{aligned} E &= \sum_{i=1}^{k} \left(n_i \cdot \sum_{x,y \in C_i} \frac{rocklink(x,y)}{n_i^{1+2f(\theta)}} \right) \\ &= \sum_{i=1}^{k} \left(n_i^{-2f(\theta)} \cdot \sum_{x,y \in C_i} rocklink(x,y) \right), \end{aligned}$$

where $n_i = |C_i|$ is the number of points contained in cluster i. The choice of the function $f(\theta)$ depends on the underlying data and the kind of clusters one is interested in. It is proposed as $f(\theta) = \frac{1-\theta}{1+\theta}$. The derivation of this function is described in detail in Guha et al. (2000). The numbers n_i of points in the clusters i are included in the objection function in order to prevent a clustering in which all points are assigned to one single cluster. The function $f(\theta)$ has the following property. Each point in cluster C_i has approximately $n_i^{f(\theta)}$ neighbors in C_i. If such a function does exist, each point in cluster C_i contributes $n_i^{2f(\theta)}$ ROCK-links – one for each pair of its neighbors. Thus, $n_i^{1+2f(\theta)}$ is an estimation for the total number of ROCK-links in cluster C_i.

The goal of the algorithm is to maximize the sum of ROCK-links in the single clusters. This leads to a minimization of the number of ROCK-links in different clusters.

Goodness Measure

The objective function described above is used to estimate the goodness of clusters. In order to determine the best pair of clusters (C_i, C_j) to be merged at each step of the algorithm, a goodness measure is introduced:

$$g(C_i, C_j) = \frac{rocklink\,[C_i, C_j]}{n^{exp}(i, j)},$$

where the number of ROCK-links between two clusters is calculated as sum of ROCK-links between the points of different clusters

$$rocklink\,[C_i, C_j] = \sum_{\substack{x \in C_i \\ y \in C_j}} rocklink\,(x, y)$$

and a weighting function $n^{exp} : K \times K \to \mathbb{R}^+$ is introduced.

Intuitively, it seems reasonable to merge in each step those two clusters with the largest number of connecting ROCK-links between each other. This may work for well-separated clusters, but in case of outliers or clusters with points that are neighbors, a large cluster may absorb other clusters because a large cluster typically has more links with other clusters. For this reason, the ROCK-link value of the goodness measure is divided by the difference of the expected number $n^{exp}(i, j)$ of ROCK-links after merging the clusters C_i and C_j. Guha et al. (2000) propose

$$n^{exp}(i, j) = (n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}.$$

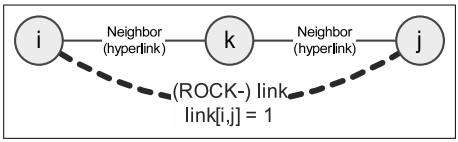


Figure 3.4: ROCK-link calculation for undirected graph

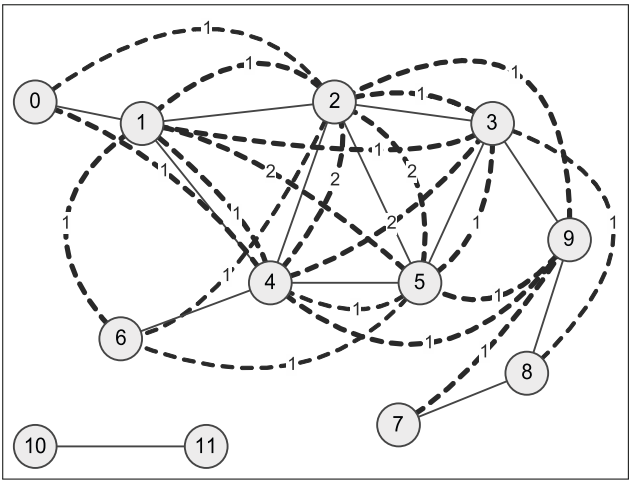


Figure 3.5: Undirected graph example with ROCK-link values

The expected change of the number of ROCK-links consists of the expected ROCK-link number of the newly merged cluster minus the expected ROCK-link numbers of the former clusters.

Computation of ROCK-links

In contrast to the original algorithm, the ROCK-links do not have to be created using the similarity threshold θ when dealing with hyperlinks in a Web graph. For the Web clustering, hyperlinks can be used as a basis for ROCK-links like shown in Figure 3.4. In a simple definition neglecting the direction of hyperlinks, two webpages i and k are neighbors, if there is a hyperlink pointing from i to k or from k to i. If the webpages i and j have only k as a neighbor, then $rocklink(i, j) = 1$.

A sample graph in Figure 3.5 illustrates the ROCK-link calculation. Webpages 1 and 5 have the two webpages 2 and 4 as common neighbors (in the graph marked

Algorithm 3.1 ROCK-link computation (Guha et al., 2000)

```
procedure compute_links(S)
  compute nbrlist[i] for every point i in S
  set rocklink[i, j] to be zero for all i, j
  for i := 1 to |S| do {
    N := nbrlist[i]
    for j := 1 to |N| − 1 do
      for l := j + 1 to |N| do
        rocklink[N[j], N[l]] := rocklink[N[j], N[l]] + 1
  }
```

with solid lines), and thus $rocklink(1,5) = 2$ (in the graph marked with a dashed line labeled with the number of ROCK-links).

Even though the threshold θ is not needed for a comparison with the similarity function, a value of θ is used in the goodness function. For the following calculations, it will be set to $\theta := 0.5$ which results in $f(\theta) = \frac{1-\theta}{1+\theta} = \frac{1}{3}$, and thus $1 + 2f(\theta) = \frac{5}{3}$.

Algorithm 3.1 describes the calculation of ROCK-link values. It consists of the procedure `compute_links` that starts with the initialization of all link values to $rocklink[i, j] = 0$. For each webpage i in the set of webpages S it selects its neighbors $N = nbrlist[i]$. Each time two common neighbors j and l of i are identified, their ROCK-link value is increased by one. The complete matrix of ROCK-links after the calculation based on the sample graph is shown in Figure 3.6.

Algorithm

The ROCK algorithm (Algorithm 3.2) starts the identification of clusters with the calculation of ROCK-link values. Each vertex s of the set of vertices S is regarded as an initial cluster. The procedure `compute_links` calculates the number of ROCK-links for every pair of clusters or vertices in S. Based on these values, the procedure `build_local_heap` constructs a heap $q[s]$ for every cluster $s \in S$. Each local heap $q[i]$ is filled with every cluster C_j with $rocklink[i, j] > 0$ in decreasing order of their goodness $g(C_i, C_j)$. A global heap Q containing all clusters is build by the procedure `build_global_heap`. Here, the clusters C_i are stored in decreasing order of their best goodness measure contained in the corresponding local heap $q[i]$.

i \ j	0	1	2	3	4	5	6	7	8	9	10	11
0			1		1							
1			1	1	1	2	1					
2	1	1		1	2	2	1			1		
3		1	1		2	1			1			
4	1	1	2	2		1						
5		2	2	1	1		1			1		
6		1	1			1						
7										1		
8				1								
9			1			1		1				
10												
11												

Figure 3.6: Matrix of ROCK-links for undirected graph

The following loop is repeatedly processed as long as the global heap contains more than the desired output number k of clusters. The cluster u with the highest goodness value (with any other cluster) is chosen from the global heap. The cluster v with the highest goodness value (with cluster u) is chosen from the local heap $q[u]$. The clusters u and v are merged into a new cluster w. This operation leads to the maximum possible increase of the criterion function in this step. The rocklink array is updated for the new cluster w with all involved clusters, u and v are deleted from the affected local heaps and w is inserted in the local heaps. The local heap Q is updated and the new cluster w is inserted to Q together with a new local heap $q[w]$.

The result of the ROCK algorithm consists of k clusters. Running the loop $|S|$ times leads to one remaining cluster containing all original vertices. k clusters need $(|S|-k)$ iterations. The tree in Figure 3.7 shows the results after each iteration for the undirected graph example. The global heap Q contains for example after the fourth iteration the clusters $\{0,1,2,3,6,7\}$, of which the clusters $\{1,2,3,7\}$ consist of two vertices each and are named by one of their merged clusters. One can observe that the strongly linked clusters $\{1,5,2,4\}$ are grouped together first. The webpages 10 and 11 will remain in singleton clusters because they do not possess enough links and are thus omitted in the solution tree.

Algorithm 3.2 ROCK clustering algorithm (Guha et al., 2000)

```
procedure CLUSTER(S,k)
  rocklink:=compute_links(S)
  for each s ∈ S do
    q[s] := build_local_heap(rocklink,s)
  Q := build_global_heap(S,q)
  while size(Q) > k do {
    u := extract_max(Q)
    if u = 0 then stop
    v := max(q[u])
    delete(Q,v)
    w := merge(u,v)
    for each x ∈ q[u] ∪ q[v] do {
      rocklink[x,w] := rocklink[x,u] + rocklink[x,v]
      delete(q[x],u); delete(q[x],v)
      insert(q[x],w,g(x;w)); insert(q[w],x,g(x,w))
      update(Q,x,q[x])
    }
    insert(Q,w,q[w])
    deallocate(q[u]); deallocate(q[v])
  }
```

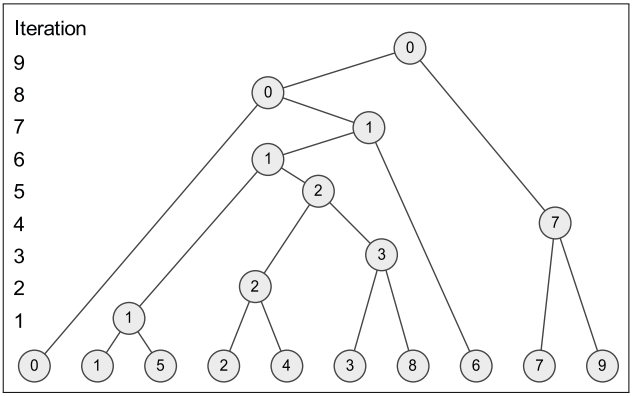


Figure 3.7: Solution tree after nine iterations for undirected graph example

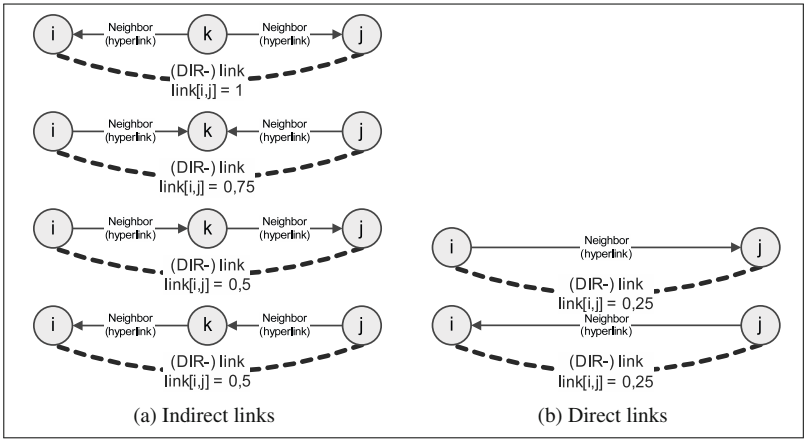


Figure 3.8: ROCK-link calculation for directed graph

3.4.2 Inclusion of Link Direction

So far, the ROCK-link calculation neglects the direction of the hyperlinks. In the following, the ROCK-link calculation is extended by the capability to consider directed links. Four cases of directed links that connect two webpages via a third one are listed in Figure 3.8a. If a webpage k contains hyperlinks to webpages i and j ($i \leftarrow k \rightarrow j$), their ROCK-link value will be increased to

$$rocklink[i,j] = rocklink[i,j] + 1$$

because i and j may deal with the same topic with a high probability. If two pages i and j have links to the same webpage k ($i \rightarrow k \leftarrow j$), they are assumed to deal with the same topic with a smaller probability and receive a value of

$$rocklink[i,j] = rocklink[i,j] + 0.75.$$

If webpage i links only indirectly to j by linking to k which again links to j ($i \rightarrow k \rightarrow j$) a value of

$$rocklink[i,j] = rocklink[i,j] + 0.5$$

Algorithm 3.3 Extended ROCK-link calculation

```
procedure compute_links(S)
  compute nbrlist[i] for every point i in S
  set rocklink[i, j] to be zero for all i, j
  for all k do {
    for all i ∈ succ(k) do {
      for all j ∈ succ(k)\i
        rocklink[i, j] := rocklink[i, j] + 1        // i ← k → j
      for all j ∈ pred(k)\i
        rocklink[j, i] := rocklink[i, j] + 0.5      // i ← k ← j
      rocklink[i, k] := rocklink[i, k] + 0.25       // k → i
    }
    for all i ∈ pred(k) do
      for all j ∈ pred(k) do
        rocklink[i, j] = rocklink[i, j] + 0.75      // i → k ← j
  }
```

is set. The same value will be added, if i is only indirectly linked by j ($i \leftarrow k \leftarrow j$). For direct links ($i \rightarrow j$) a small value

$$rocklink[i, j] = rocklink[i, j] + 0.25$$

is added to take into account the original hyperlinks, but not to weight them too much. The ROCK-link values for direct links are illustrated in Figure 3.8b.

The calculation of ROCK-link values under consideration of link directions is performed by an extended `compute_links` procedure, shown in Algorithm 3.3. It replaces Algorithm 3.1 and assigns different weights for the four cases of indirect links (Figure 3.8a) and for the direct links (Figure 3.8b). The algorithm uses the successor set $succ(i)$ that contains all webpages the page i is linking to. The predecessor set $pred(i)$ consists of those webpages that contain a link to page i.

The sample directed graph in Figure 3.9 leads to the solution tree shown in Figure 3.10. Comparing this graph with the solution shown in Figure 3.7 reveals the influence of the link directions on the clustering process. Vertex 5 remains the first one to be clustered, but because of the link directions it is merged with vertex 2 instead of 1. The clusters 1 and 6 are merged earlier in the process because a common predecessor (in this case 4) adds a relatively high ROCK-link value. The solution tree of the directed graph example shows a different order of inclusion

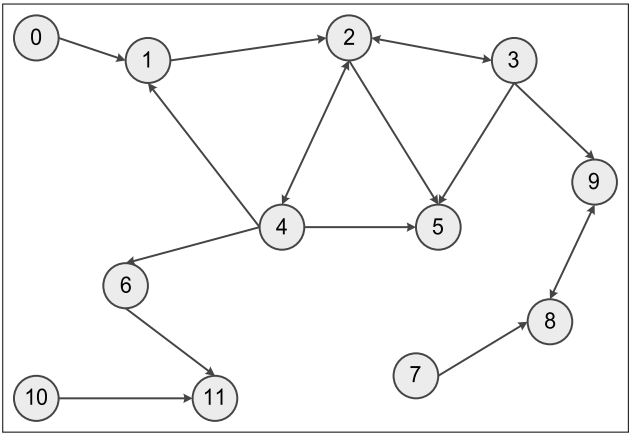


Figure 3.9: Sample directed graph

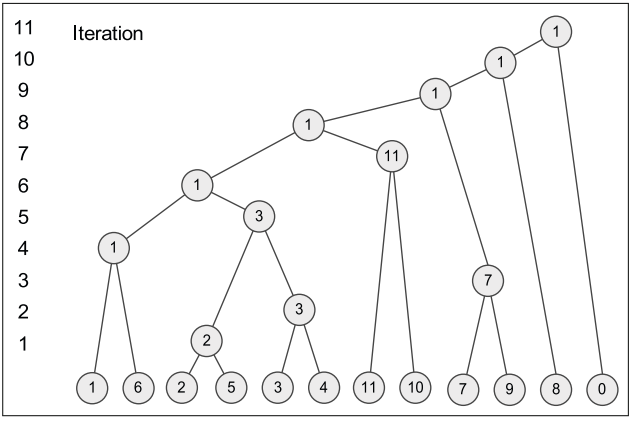


Figure 3.10: Solution tree for directed graph example

of vertices into clusters and thus results into different clustering solutions after completing the iterations.

3.4.3 Architecture

The ROCK algorithm is applied on Web-search results using the following architecture. The search-result and link data for a specified query is gathered from an

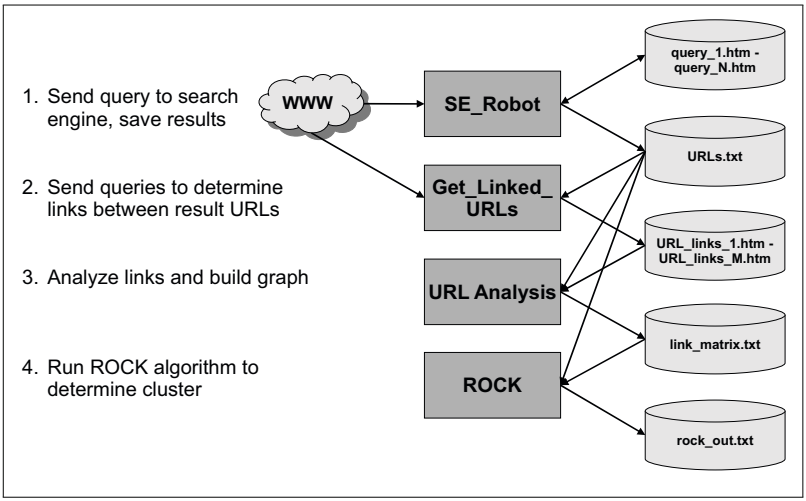


Figure 3.11: Architecture for running ROCK on Web query results

existing Web-search engine. The ROCK algorithm runs on a graph built from the URL information returned. Four different modules that store their intermediate results in different files (shown in Figure 3.11) perform the process.

The first module `SE_Robot` sends the specified search query to a publicly available search engine. The number of results delivered on one page can be increased, but it is limited by the search-engine provider, often to 100 results. Thus, multiple different requests to the search engine are necessary in order to gain enough results for a Web graph of sufficient size. The result pages returned are stored in files named `query_1.htm` until `query_`*N*`.htm` (with the total number of result pages *N*). In a second step, the `SE_Robot` module extracts all URLs from the result pages and stores them in the file `URLs.txt`. This file is the input for the next module.

The module `Get_Linked_URLs` generates queries based on the URLs from `URLs.txt`. The queries are sent to the search engine in order to request linkage information. Again, the search engine's result pages are stored in files. For each URL, one link file is generated (`URL_links_1.htm` until `URL_links_`*M*`.htm`, for *M* URLs).

The module `URL_Analysis` generates a link matrix from the URL-file and the link files and stores it in the `link_matrix.txt` file. This file serves as basis for the `ROCK` module, which calculates the distances (ROCK-links) and runs the

clustering algorithm in order to determine the clusters. The URL-names for the cluster members which were not part of the link matrix are recovered from the URL file and are assigned to the output clusters. The results are stored in the file `rock_out.txt`. Besides the clusters, this file contains statistical data concerning the clustering process like the number of hyperlinks (neighbors), ROCK-links and the link-degree.

The architecture was implemented in C++ on a 3.6 GHz P4 personal computer using MICROSOFT VISUAL STUDIO .NET. The results returned by this architecture are analyzed in the next subsection.

3.4.4 Experimental Results

The ROCK algorithm is tested using search results of GOOGLE. GOOGLE is able to return up to 1'100 results per query even though it often states to dispose of many more results. If available, up to 1'100 links were gathered by the `Get_Linked_URLs` module for each of the result URLs.

A sample output for the query "computer" consisting of eleven clusters with four URLs each is exemplarily shown in Table 3.1. For the sake of clarity, only the first clusters generated by the ROCK algorithm are shown. At the first glance, the search results are composed to reasonable clusters. However, the interpretation of the clusters is not obvious because of the lack of a helpful description. There is no straight-forward approach for the generation of cluster descriptions based on hyperlink data. This problem can be solved by including textual data in the clustering process. A method for obtaining clusters including descriptions will be presented in the Section 3.5.

Another problem of the purely link based approach follows from the densities of the link matrices. Table 3.2 shows metrics for link matrices of different sizes for a typical query. In a matrix of $|S| = 100$ pages, there were only eleven hyperlinks resulting in six ROCK-links. Running the ROCK algorithm without limiting the cluster number k, 95 clusters are created. That means that there is a maximum of $100 - 95 = 5$ clusters that do consist of more than one webpage. Webpages that neither contain inlinks nor outlinks to other webpages in the result set are not added to any cluster. The link degree calculated by the number of neighbors divided by the number of pages in this matrix is at 0.11. A low link degree limits the merging process of clusters.

1:	http://www.computer.org/;
3:	http://www.computer.org/computer;
333:	http://www.computer.org/careers;
866:	http://www.computer.org/computer/about.htm;
229:	http://www.december.com/cmc/mag;
332:	http://www.december.com/cmc/info;
41:	http://www.computer.com/;
157:	http://www.computer.is/;
983:	http://www.old-computers.com/;
1078:	http://www.ocm.com/;
450:	http://www.cambridge-computers.com/;
978:	http://www.ocm.com/;
945:	http://www.sarc.com/;
1045:	http://www.sarc.com/;
2:	http://www.cai.com/;
15:	http://www.cert.org/;
956:	http://www.dell.com/jp;
1022:	http://www.secureroot.com/;
12:	http://www.planet.nl/computer;
84:	http://hoaxbusters.ciac.org/;
436:	http://www.eicar.org/;
727:	http://www.mcafee.com/;
233:	http://antivirus.cai.com/;
361:	http://results.about.com/computer;
740:	http://web.planet.nl/computer;
922:	http://www.secureroot.com/;
139:	http://www.yahoo.com/Science/Computer_Science;
814:	http://www.computer-consulting.com/;
6:	http://www.computer.de/;
517:	http://www.computer-computer.de/;
150:	http://www.computer-archiv.de/;
314:	http://www.internet-verzeichnis.de/computer&software;
97:	http://www.computer.shops-here.com/;
135:	http://www.computer-woerterbuch.de/;
196:	http://www.supportnet.de/;
419:	http://www.glossar.de/glossar/z_computer.htm;
498:	http://directory.google.com/Top/World/Deutsch/Computer;
882:	http://www.yahoo.com/Computers_and_Internet/Security_...
204:	http://www.computer-dating.com/;
688:	http://www.comp-connection.com/;
142:	http://www.computer-expo.ch/;
163:	http://www.bcs.org.uk/;
800:	http://www.cs.umass.edu/;
810:	http://www.cs.brown.edu/;

Table 3.1: First clusters for the query “computer”

Increasing the number of pages up to 4'000 in the link matrix, the link degree grows very slowly and reaches only a value of 0.47, which corresponds to only 1'869 links in the 4'000×4'000 matrix. The sparseness of the matrix is illustrated in Figure 3.12, where every symbol[2] corresponds to a hyperlink from an URL plotted on the abscissa to an URL plotted on the ordinate.

The next section describes a combined approach that tries to overcome the problems of the purely link based approach.

3.5 Combined Approaches

In this section, we introduce an approach combining text based and link based methods (Lieberam-Schmidt and Pesch, 2008). This approach aims at compensating the weaknesses identified for purely text based and purely link based algorithms. Several clustering methods will be tested and analyzed using the same framework based on textual and link information.

3.5.1 Architecture

A common architecture is used to implement the different combinations of clustering approaches. The architecture is explained along the information flow shown in Figure 3.13. The user triggers the process by sending a query to the system. Results from publicly available search engines (external result page) are gathered and stored.

[2]Here, the shape and the color of the symbols are not relevant.

Number of Pages	Number of Neighbors (Hyperlinks)	Number of ROCK-links	Number of Clusters	Link-Degree
100	11	6	95	0.11
500	145	424	413	0.29
1'000	318	930	782	0.32
2'000	811	3'276	1'549	0.41
3'000	1'377	1'388	2'252	0.46
4'000	1'869	13'637	3'023	0.47

Table 3.2: ROCK statistics

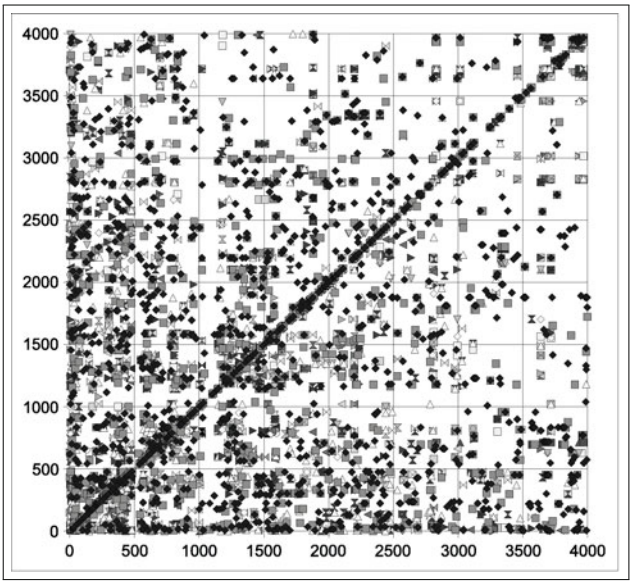


Figure 3.12: Link density

The following preprocessing steps prepare the data for the clustering process. The steps' elements are described in the next subsection. The different clustering methods use the same preprocessing. Depending on the clustering method used, different distance measures and document representations are required. The document representation is explained in Subsection 3.5.3.

After the calculation of distance values, one of the clustering methods described in Subsection 3.5.4 is performed. Depending on the method, either the cluster description is derived from the determined cluster elements, or suitable descriptions are determined before the cluster elements are assigned to them. All methods return their results to the user on the internal result page.

3.5.2 Preprocessing

At first, a data basis of Web search results is created and appropriately stored after the data gathering and preparation. Three preprocessing steps are performed before the clustering process can be started. The lower left box in Figure 3.13 contains the following steps that will be explained in this subsection:

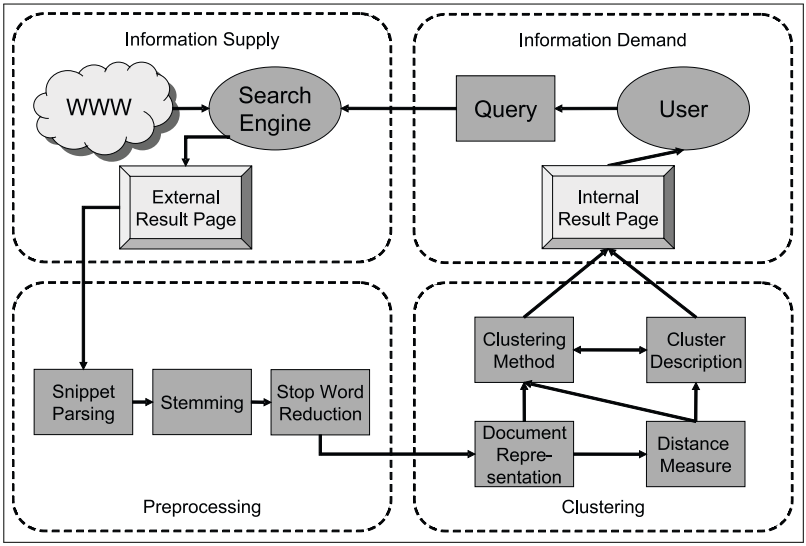


Figure 3.13: Clustering architecture – information flow

1. Snippet Parsing
2. Stemming
3. Stop Word Reduction

Snippet Parsing

In order to build a data basis, a search query is placed at a publicly available search engine (for this work we chose GOOGLE and LOOKSMART[3]) and it stores the results. The procedure described in Subsection 2.6.1 is used for gathering link information. The gained amount of webpages is called *root set.* The root set can be extended to a base set. The extension helps to include additional documents which might be relevant but which would otherwise not have been considered. A non-consideration can result from documents that were not found by the search engine, as they do not contain the keywords of the query even if they would thematically fit in the search query. Data structures for storing the link matrices and snippets of the different sets are described in Section 2.5. Two ways to create the base set are introduced in the following.

[3] `www.google.com` and `www.looksmart.com`

The first approach is oriented at the one of the HITS algorithm *(Hyperlink Induced Topic Search;* Kleinberg, 1999). The HITS algorithm differentiates webpages not only concerning their relevance to a search query *(authority)*, but additionally considers their quality as a guide *(hub)* to relevant Web pages (compare Subsection 2.4.2). Kleinberg starts with a root set of approximately 100 – 200 webpages that are yielded as a result to a query by an existing search engine. This root set does not necessarily contain all pages relevant for the search term. However, with a certain probability these sites have hyperlinks to or from relevant pages. Therefore, two kinds of pages are determined in a second step. First, pages which contain links to pages of the root set and second, those pages the pages of the root point to. In the HITS algorithm, approximately 50 hyperlinks per page are considered so that the extended set, the so-called *base set,* normally consists of 1'000 to 5'000 webpages. For each page included in the base set the HITS algorithm determines authority and hub values. The generation of a base set is of particular importance for this work in order to cluster documents on its basis.

The base-set generation of the HITS algorithm can lead to a so-called *topic drift* – in particular with precisely formulated search queries. Besides, the thematic focus of the content of documents can shift to a more general context that mostly covers the particular query context as a subset. We suggest the *Aroundlink Algorithm (ALA),* which is oriented at the *Cocitation Algorithm* of Dean and Henzinger (1999) as an alternative means for gaining of document sets.

The starting point of the aroundlink algorithm is – as it is with the HITS algorithm – a root set containing 100 – 200 documents. But the extension of the amount is being carried out in another way: For a number of documents – determined by the user – that contain hyperlinks (inlinks) which point to documents of the root set, hyperlinks are added as follows. Those hyperlinks that are located in the HTML-text in proximity of the inlinks extend the root set in the sequence of their appearance. This proceeding is based on the assumption that many hubs contain links that are assorted by topics, whereby links to documents with similar content can be found in proximity to each other. In this way, we can avoid – with respect to extensive hubs – the admission of links that are thematically located far away from the search context. The documents of the inlinks itself are not admitted into the document set but are only used as a source of information.

Stemming and Stop Word reduction

After extension of the root set, two preparing steps, i.e. stop-word elimination and stemming, are carried out in order to more precisely describe the semantic content of the collected documents. Those words that do not possess any direct connection with the topic of the text, e.g. articles, pronouns and prepositions, are called *stop-*

words. In order to avoid these stop words from being included into the similarity measurement of the documents they are eliminated from the data basis. This can either be done after completion of the information acquisition or directly during the admission of the text.

In a further step, words of the same root are tracked back to their common origin. Four stemming procedures (Frakes, 1992) are examined:

Using the *table lookup method,* the term is being searched in a global table and is being traced back to the origin registered in the table. For this purpose, the precondition is an elaborated and extensive table containing all words of a language.

The aim of the *successor variety* procedure is to isolate morphemes of a term. Morphemes are the smallest, frequently appearing logic and inseparable character strings of a language. The successor variety of a substring of a word is calculated by counting the number of different characters that follow it in words in the document set. A precipitous increase of the successor variety indicates a morpheme end.

The *n-grams* method looks at small character strings included in a word. Typically, two or three consecutive letters are used called digrams or trigrams. A string with n characters is called n-gram. The number of unique n-grams two words have in common is used as similarity measure. Words whose similarity exceeds a certain threshold are considered to be of the same stem.

In the *suffix removal* procedure, known word-endings (suffixes) of a language are removed according to defined rules and so the word is reduced to its infinitive (Lovins, 1968). The algorithm searches in an iterative procedure the longest character string (suffix) at the word's end which meets these rules and removes them. The algorithm of Porter (1980) is often used because of its compact structure and small need of computing time. The Porter algorithm examines three types of conditions concerning the (determined) origin, the kind of word ending and the replacements already made. Afterwards, it carries out one or more replacement rules: By placing minimum requirements to the quality of the resulting word origin, it prevents a reduction to too short stems. For this reason, the algorithm evaluates the origin with a measure counting the number of alternating vowel consonant sequences.

While running times of the successor variety and the n-grams method depend on the whole number of words and the running time of table lookup depends on the size of the word-table, the running time of the Porter algorithm is determined only by a double-digit number of rules. Due to the favorable characteristics of

running time and the easy implementation with qualitatively good results, we use the Porter algorithm for this research.

3.5.3 Document Representation

The next issue to regard is the format of document representation. It determines the data access possibilities and the functions that can be carried out on the documents.

Vector Space Model based on Term Frequency

The *vector space model* (VSM; Salton, 1989) represents text documents as vectors in the Euclidean space. Each term of a document corresponds to one vector dimension. The text of a search query can be modeled as a vector in the same way.

A term-document matrix $A = (a_{ik})_{1 \leq i \leq N, 1 \leq k \leq M}$ represents the document set. Each entry $a_{ik} \in \mathbb{R}_0^+$ states the relative contribution of a term i ($1 \leq i \leq N$) to the representation of the document k ($1 \leq k \leq M$). The matrix entry a_{ik} can – in the simple case – reflect the state if the term at all exists in the document *(binary weighting),* or the number of its appearances in the document (*term frequency,* tf).

Let $f_{ik} \in \mathbb{N}$ be the number of occurrences of term i in document k, then the matrix entry for binary weighting is defined as

$$a_{ik}^{\text{binary}} = \begin{cases} 1 & \text{if } f_{ik} > 0 \\ 0 & \text{otherwise} \end{cases}.$$

Under the assumption that a term has a higher impact on the content the more often it appears in the document, the matrix can be defined by

$$a_{ik}^{\text{tf}} = f_{ik}.$$

In order to consider furthermore that one term the more discriminates documents from each other the less it exists in other documents, the term frequency is weighted with the inverse frequency of these documents *(inverse document frequency, idf)* in which this term exists *(tf-idf)*:

$$a_{ik}^{\text{tfidf}} = f_{ik} \log\left(\frac{N}{df_i}\right),$$

where df_i is the number of documents with $f_{ik} > 0$. Hereby, words appearing in very many documents receive a smaller weight than seldom occurring words. The logarithm function smoothes the effect of very rare words.

These values can be normalized using the L_2-norm as follows in order to prevent large documents from dominating the document space:

$$a_{ik} = \frac{f_{ik} \log\left(\frac{N}{df_i}\right)}{\sqrt{\sum_{k'=1}^{M} \left(f_{ik'} \log\left(\frac{N}{df_i}\right)\right)^2}} = \frac{f_{ik}}{\sqrt{\sum_{k'=1}^{M} \left(f_{ik'}\right)^2}}.$$

In order to compare documents to each other, similarity or distance measures are needed. The cosine coefficient $\cos\gamma$ is often used as a similarity measure because it can be intuitively interpreted as an angle γ between two vectors in the underlying vector space. Moreover, it is easy to calculate. The angle between two vectors v_1 and v_2 is determined with

$$\cos\gamma = \frac{\langle v_1, v_2 \rangle}{\|v_1\| \cdot \|v_2\|},$$

where $\langle v_1, v_2 \rangle$ is the standard scalar product of the vectors and $\|v_1\|$ the Euclidean norm of vector v_1.

Thus, the similarity of two documents d_h and d_j can be calculated using the cosine coefficient as

$$\mathrm{sim}_{cos}^{tfidf}(d_h, d_j) = \frac{\sum_{i=1}^{N} f_{ih} \log\left(\frac{N}{df_i}\right) f_{ij} \log\left(\frac{N}{df_i}\right)}{\sqrt{\sum_{i=1}^{N} \left(f_{ih} \log\left(\frac{N}{df_i}\right)\right)^2} \sqrt{\sum_{i=1}^{N} \left(f_{ij} \log\left(\frac{N}{df_i}\right)\right)^2}}.$$

If the tf-idf vectors d_h, d_j exist already in the above described normalized form, the calculation can be performed as scalar product

$$\mathrm{sim}_{cos}^{tfidf}(d_h, d_j) = \langle d_h, d_j \rangle.$$

The words of a search query can be presented as a vector (like a document). Thus, best suitable documents for a query can be determined by using this similarity measure to compare a query vector with the document vectors in the same vector space.

In the vector space model exclusively those words are regarded to be equal that are literally equal. Furthermore, the VSM contains the implicit assumption that the words of a document appear independently from each other. Normally, this is not the case as the words have a common context. Without considering the context of single words, redundant information would be stored in the term-document matrix. Not considering the context leads to the fact that different words with identical meaning *(synonyms)* cannot be found. Moreover, identically spelled words with different meanings *(polysemes)* in this case cannot be differentiated by looking at the context.

Other applications of the vector space model are e.g. presented by Aasheim and Koehler (2006) who have used the VSM for classifying the content of Web documents.

Latent Semantic Indexing

Latent Semantic Indexing (LSI; Deerwester et al., 1990) tries to avoid the known problems of the VSM by considering *concepts* instead of single words. Thereby, it is assumed that the words used in a document form an *intrinsic* or *latent structure,* which is concealed by the variability of selection of words. In order to identify the concepts existing behind this structure, a *Singular Value Decomposition (SVD)* is being carried out on the term-document-matrix A. The matrix A can be decomposed in the following matrices, because any rectangular matrix can be decomposed into the product of three other matrices (compare Deerwester et al., 1990):

$$A = U\Sigma V^T.$$

The columns of matrix U contain the orthogonal eigenvectors of matrix AA^T and the columns of matrix V contain the orthogonal eigenvectors of A^TA. The values on the diagonal of the matrix

$$\Sigma = diag(\sigma_1 ... \sigma_r)$$

are called *singular values* σ_i. There are $r = rang(A)$ singular values. The smaller the singular values are the more similar are the corresponding vectors. By discarding some of the singular values, the size of the basis can be reduced.

If we only keep the k biggest singular values in the matrix Σ and call this matrix Σ_k, this will lead to a k-rank approximation. Figure 3.14 shows the principle of the k-rank approximation for $k = 2$. The resulting matrix

$$A_k = U\Sigma_k V^T$$

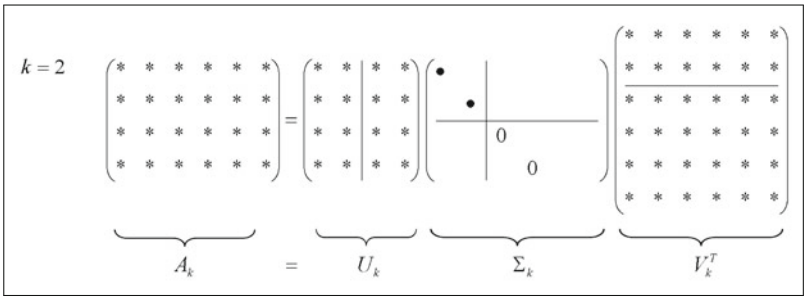


Figure 3.14: k-rank-approximation (Osinski, 2003)

presents the best approximation of matrix A for a given k using the L_2-norm. The approximation A_k tries to keep as much information of A as possible for a given k. By keeping only the highest singular values, those singular values are kept that describe best the relationship between terms and documents. If two documents do not differ very much from each other, they probably belong to the same concept. That means that these documents are only separated by a lower singular value. Removing the smaller singular values may show that these documents belong to a common concept.

Thus, documents containing many common terms – and therefore probably a similar concept – get a similar representation in the vector area. The number of concepts to be extracted can be arbitrarily chosen by setting a value for k. The concepts of the SVD can be used as basis for clustering. Moreover, search results are getting more robust against different spellings and polysemy of the search terms.

3.5.4 Clustering Method Combination

In the following, different methods of clustering implemented in the context of this work are described. First, the *Probabilistic Latent Semantic Indexing* model *(PLSI)* and the *Probabilistic HITS* model *(PHITS)* are introduced. Both models are based on probability processes. Where in the PLSI model (in a similar way like with LSI) words represent a document, in the PHITS model the link-structures of the document-set (like in the HITS model) are examined.

The *PLSI-PHITS Clustering* approach *(PPC)* combines these two models. It generates descriptions fitting to the latent classes and builds clusters from documents matching these descriptions. In Figure 3.15, the components of PPC are connected with a dashed line.

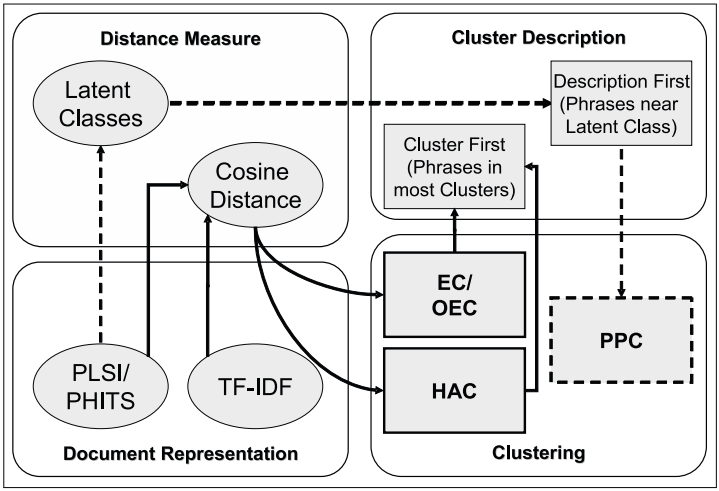


Figure 3.15: Method combination

The *hierarchical agglomerative clustering* (HAC) and the *Ejection Chain* heuristics (EC and OEC) first build clusters based on cosine distances based on PLSI/PHITS and tf-idf and then generate phrases that best match these clusters. In Figure 3.15, the components of HAC and EC/OEC are connected with a solid line.

This subsection explains the elemental methods and their combination resulting in cluster procedures as shown in Figure 3.15.

Probabilistic Latent Semantic Indexing Model (PLSI)

The *Probabilistic Latent Semantic Indexing Model* (PLSI; Hofmann, 1999a,b) is based on a statistical approach that was published under the name *Aspect Model* and *Aggregate Markov Model (Saul and Pereira, 1997).* The goal of the PLSI model is to identify the abstract concepts in a document collection. For this reason, the creation of the documents

$$d \in D = \{d_1, ..., d_N\}$$

from the words

$$w \in W = \{w_1, ..., w_M\}$$

is regarded as a generative, statistical process. Thereby, class variables

$$z \in Z = \{z_1, ..., z_K\}$$

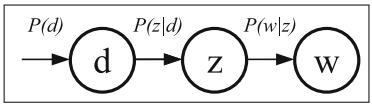


Figure 3.16: Generative process

are introduced through which the appearance of a word w in a document d is explained. Class variables can be interpreted as different topics or clusters, which are assigned to documents based of the words they contain. The class variables correspond to the abstract concepts and the first k singular-values of the k-rank approximation in the LSI procedure.

The goal of the PLSI is to extract the abstract concepts defined by latent classes. For this reason, we assume that a generative process exists that creates words from latent classes that are created from documents. We further assume, that the generative process can be described with probability functions. The probability functions are not known in advance, but their determination helps to identify latent classes, as described in the following.

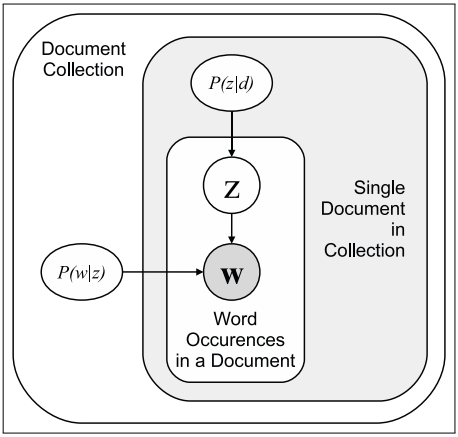


Figure 3.17: PLSI – graphical model (Hofmann, 2001)

The generative process consists of three steps (compare Figure 3.16):

1. Choose a document d with the probability $P(d)$.
2. Choose a latent class z for a document d with the probability $P(z|d)$.
3. Create a word w for a latent class z with the probability $P(w|z)$.

On the basis of this generative process a probability distribution for the common appearance of a document d and a word w can be described as follows:

$$P(d,w) = P(d) \cdot P(w|d)$$

with

$$P(w|d) = \sum_{z \in Z} P(w|z)P(z|d).$$

Thus, we assume the independent creation of all document-word pairs as well as the independence of a word and a document from the state of a given class variable.

The relationship among the different sets involved and the probabilities can be explained by means of Figure 3.17. The outer box represents the whole document collection D. In a single document of the collection (represented by the middle box), the occurrences of a word are represented by the inner box. The generation

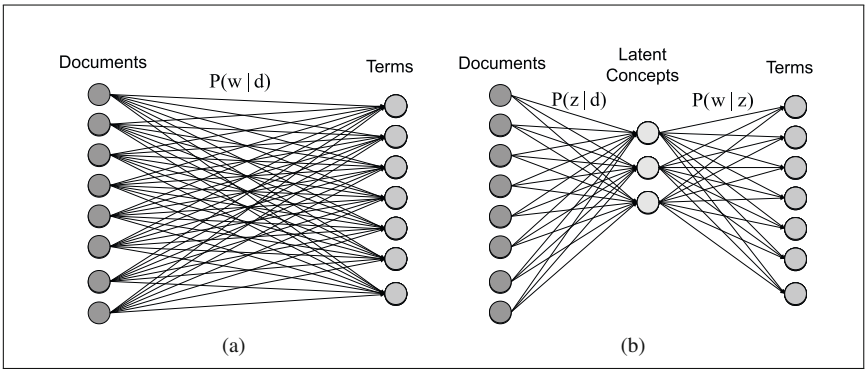


Figure 3.18: PLSI bottleneck (Hofmann, 2001)

of a word w out of a latent class z can be described by two probabilities. $P(z|d)$ is the same for all words in the document. $P(w|z)$ is the same for all documents in the collection. Summing up the product of both probabilities for all $z \in Z$ leads to the calculation of $P(w|d)$.

The introduction of the latent classes z can be interpreted as a bottleneck (compare Figure 3.18). The number of probabilities involved reduces from $|D| \cdot (|W|-1) = N \cdot (M-1)$ (Figure 3.18a) to $|D| \cdot (|Z|-1) + |Z| \cdot (|W|-1) = N \cdot (K-1) + K \cdot (W-1)$ (Figure 3.18b).

Now the probability distributions $P(d)$, $P(z|d)$ and $P(w|z)$ for a given document set D with words W as well as a given number K of class variables z_i are searched for. This is performed by the maximization of the log-likelihood function

$$L = \sum_{d \in D} \sum_{w \in W} n(d,w) \log P(d,w),$$

where $n(d,w)$ denotes the number of the common appearance of the word w and document d:

$$n(d,w) = \left| \left\{ (i,j) \,|\, i \in \{1,...,N\} \wedge j \in \{1,...,N\} \wedge d_i = d \wedge w_j = w \right\} \right|.$$

For the estimation of parameters we use a numerical procedure, the *Expectation Maximization (EM) algorithm* (Dempster et al., 1977), as the log-likelihood function is difficult to be solved analytically. The EM algorithm finds approximations for the probabilities of class variables z by first determining a lower bound for the log-likelihood function and approaching the probability distribution in a two-step

iterative procedure. The approximation is split into two steps, the *expectation step (E-step)* and the *maximization step (M-step)*. The steps are performed in alternation as follows. The E-step estimates the probabilities as

$$\begin{aligned} P(z|d,w) &= \frac{P(z,d,w)}{P(d,w)} \\ &= \frac{P(z)\,P(d,w|z)}{\sum_{z' \in Z} P(z',d,w)} \\ &= \frac{P(z)\,P(d|z)\,P(w|z)}{\sum_{z' \in Z} P(z')\,P(d|z')\,P(w|z')} \end{aligned}$$

and the M-step estimates the probabilities as

$$P(w|z) = \frac{\sum_{d \in D} n(d,w)\,P(z|d,w)}{\sum_{d \in D} \sum_{w' \in W} n(d,w')P(z|d,w')},$$

$$P(d|z) = \frac{\sum_{w \in W} n(d,w)P(z|d,w)}{\sum_{d' \in D} \sum_{w' \in W} n(d',w')P(z|d',w')},$$

and

$$P(z) = \frac{\sum_{d \in D} \sum_{w \in W} n(d,w)P(z|d,w)}{\sum_{d \in D} \sum_{w \in W} n(d,w)}.$$

$P(z|d,w)$ denominates the probability that a word w in a document d is explained by a factor corresponding to the class z. The probabilities are initialized with random values. The convergence of the EM algorithm was proven by Dempster et al. (1977). In order to avoid an overfitting to the given data – which can e.g. emerge from the existence of only a small number of non-trivial data – the algorithm is extended with techniques of *Deterministic Annealing* (Rose et al., 1990) to a *tempered EM (TEM) algorithm*. By using a parameter β $(0 < \beta < 1)$ distributions of the posterior probabilities (with decreasing β) are converged to the

equipartition:

$$P_\beta(z|d,w) = \frac{(P(z)P(d|z)P(w|z))^\beta}{\sum_{z' \in Z} (P(z')P(d|z')P(w|z'))^\beta}.$$

The parameter β is initialized with $\beta := 1$ and reduced using a factor η ($0 < \eta < 1$) in each iteration. After β has reached a certain threshold, a fixed number iterations of the (non-tempered) EM algorithm is performed. Experiments have shown, that after five iteration of the EM algorithm stable solutions are reached. The reduction of the relative influence counteracts the phenomenon of overfitting. The procedure is shown in Algorithm 3.4.

As the number of the basic class variables is not determined by the EM-algorithm, this number has to be specified separately. As the approaches for the determination of an optimal number that can be found in literature (Monte Carlo Approach (Smyth, 1996) and Penalized Likelihood (Moore, 1999)) considerably interfere with the running time, the number of resulting class variables is specified by the user.

For the demonstration of the relation between PLSI and LSI, the aspect model can be formulated in the matrix notation as follows.

Algorithm 3.4 Tempered EM algorithm (Hofmann, 1999a)

```
procedure tempered_EM(D)
  chose a random subset D* from document set D
  set D# = D\D*
  initialize β ← 1, i:=0, j:=0
  initialize probabilities P(z)_{z∈Z}, P(d#|z)_{d#∈D#,z∈Z} and
    P(w|z)_{w∈W,z∈Z} with random values
  calculate start value
    L_new = Σ_{d*∈D*} Σ_{w∈W} n(d*,w) log P(d*,w)
  do
    do
      set L = L_new, i ← i+1
      calculate probabilities of modified E-step:
        P_β(z|d#,w)_{z∈Z,d#∈D#,w∈W}
      calculate probabilities of M-step:
        P(z)_{z∈Z}, P(d#|z)_{d#∈D#,z∈Z} and P(w|z)_{w∈W,z∈Z}
      calculate new value of log-likelihood
        L_new = Σ_{d*∈D*} Σ_{w∈W} n(d*,w) log P(d*,w)
    while (L_new − L > 0 and i<max_inner_loop)
    set β ← ηβ with 0 < η < 1
  while (β ≥threshold and j<max_outer_loop)
  for i = 1 to 5 do {
    calculate probabilities of E-step:
      P(z|d#,w)_{z∈Z,d#∈D#,w∈W}
    calculate probabilities of M-step:
      P(z)_{z∈Z}, P(d#|z)_{d#∈D#,z∈Z} and P(w|z)_{w∈W,z∈Z}
  }
```

Let

$$U_K = (P(d|z))_{d\in D, z\in Z}$$

and

$$V_K = (P(w|z))_{w\in W, z\in Z}$$

and

$$\Sigma_K = \mathrm{diag}\,(P(z)_{z\in Z}),$$

where diag$(\cdot)$ describes a diagonal matrix with $\Sigma_K(i,j) = 0$ for $i \neq j$. Then, the word-document matrix

$$A_K = (P(d,w))_{d \in D, w \in W}$$

can be written as

$$A_K = U_K \Sigma_K V_K^T .$$

A comparison of this decomposition with the k-rank-approximation of the LSI shows that the singular values of the SVD are replaced by the probabilities of the class variables. However, both models are based on different assumptions of the probability distribution. As LSI uses the distances of the L_2-norm and thus implies a Gauss distribution, the PLSI model is based on the likelihood function that was generated by maximization of the model's prediction capability.

Now, for PLSI exist (like for the LSI) the following two applicabilities: On the one hand, latent classes can be interpreted as clusters where a value $P(w|z)$ designates the topical contribution of a word w in the concept z. On the other hand, the document representations gained can be the basis for calculating pairwise similarities between documents in order to use them for a document clustering procedure. Examples in Hofmann (1999a,b) show that the PLSI identifies meaningful topical groups in different databases for the documents contained and comes to a suitable allocation. Within the framework of our application of the PLSI model on Web-based documents, we found out, however, that some of the latent class vectors extracted contain topics of different subject areas. This is often not only a matter of very random combinations but a matter of mixtures of two or three different topics. Concerning the use on Web documents, we suggest an embedding of the PLSI procedure into a cluster-description-first approach.

In the *Suffix Tree Clustering* Algorithm (Zamir and Etzioni, 1998) phrases extracted by suffix array were used for the first time for the selection of suitable cluster descriptions. Thus, we can make sure that the descriptions of the clusters are contained in the documents of the clusters and that they counter a mixing of different subject areas or a topic drift. Zhang and Dong (2004) connected with their *Semantic Hierarchical Online Clustering (SHOC)* the LSI and the extracted document-phrases. In LINGO (Osinski et al., 2004) this procedure is combined with a cluster-description-first procedure. The cluster-description-first approach is combined with the PLSI approach in the following.

If we summarize all values $P(w|z)$ in a $M \times K$ matrix containing latent classes

$$C = (P(w|z))_{1 \leq w \leq M, 1 \leq z \leq K},$$

the K column vectors form the textual representation of the basic latent classes. In order to enable a comparison of the descriptions with the column vectors of this matrix, each phrase P_t is being interpreted as a pseudo-document $P_t = \{p_1, ..., p_M\}^T$, which shows an entity of the same word space as the documents. After evaluation of the entries of the phrase vector using tf-idf, the phrase-vector and the latent class vector are normalized regarding the L_2-norm. A matrix multiplication $M_t = P_t^T C$ yields a $K \times 1$-vector with similarity values of the phrase t to the K latent classes. A multiplication for all phrases t is yielded by the $M \times K$-matrix $M = P^T C$, which shows as column vectors the similarities between the amount of phrases and the latent class represented by the column. In order to obtain as concise cluster descriptions as possible, we now select those phrases with the highest, strongly positive similarity value that consists of at least two words and do not contain any other phrases as a part-phrase.

The allocation of the documents to the extracted cluster descriptions is carried out by assigning those documents to the cluster that exactly contain this description as a text.

Probabilistic HITS (PHITS)

Similar to the HITS algorithms, the *Probabilistic HITS Algorithm* (PHITS; Cohn and Chang, 2000) analyzes the hyperlink structure of the underlying document set. The generation of the subgraph that corresponds to the search query follows the modus operandi of the HITS algorithm. A generative process analogous to the PLSI modeling is used for the document grouping. Thus, we are also able to determine multiple authorities.

The model describes the choice of documents $d \in D = \{d_1, ..., d_N\}$ and citations $c \in C = \{c_1, ..., c_L\}$ by using class-variables $z \in Z = \{z_1, ..., z_K\}$ in three steps:

1. Choose a document d with the probability $P(d)$.
2. Choose a latent class z with the probability $P(z|d)$.
3. Create a citation c with the probability $P(c|z)$.

There, C as well as D refers to the document set. Both sets can be identical but are differentiated because of their function: C contains cited documents and D citing documents. Two assumptions are made: The creation of a document-citation pair is carried out independently of other pairs. For a given factor z, the creation of a citation and the creation of a document are carried out independently from each other.

The number of citations between c and d is denoted by

$$n(d,c) = \left|\left\{ (d_i,c_j) \middle| d_i = d \wedge c_j = c \right\}\right|.$$

The log-likelihood function

$$L = \sum_{d\in D}\sum_{c\in C} n(d,c)\log P(d,c)$$

is maximized by using the tempered EM algorithm like in PLSI. The expectation step and the maximization step are performed analogously to the PLSI procedure (Algorithm 3.4) with a control parameter β. The result can be used – as with PLSI – either directly for the clustering or for the representation in order to create a document-similarity measure for the clustering procedure. In contrast to PLSI, the complete latency class vector is used for clustering here.

PLSI-PHITS Clustering (PPC)

Web documents contain textual information as well as interconnecting hyperlinks, which – like referring citations – can express content relations between the linked documents. Therefore, we want to use both types of information in a common model. PLSI and PHITS are furthermore based on very similar modeling approaches. Thus, both models can be additively linked in a relatively easy way (here with a weighting-factor of α ($0 \leq \alpha \leq 1$)) with the following log-likelihood function (Cohn and Hofmann, 2001):

$$\begin{aligned} L \quad = \quad \sum_{d\in D} \Bigg(& \alpha \sum_{w\in W} \frac{f_{wd}\log\left(\frac{N}{n_w}\right)}{\sum_{w^*\in W} f_{w^*d}\log\left(\frac{N}{n_{w^*}}\right)} \log \sum_{z\in Z} P(w|z)P(z|d) \\ & + (1-\alpha)\sum_{c\in C} \frac{n(c,d)}{\sum_{c^*\in C} n(c^*,d)} \log \sum_{z\in Z} P(c|z)P(z|d) \Bigg). \end{aligned}$$

The generating of the underlying document set is carried out either analogously to the modus operandi of the HITS algorithm or by using the aroundlink algorithm (ALA, compare Subsection 3.5.2). The number of words as well as the number of citations is normalized per document to the document's whole number of words or citations, respectively, in order to consider each document with the same weighting. Instead of the tf-evaluation in Cohn and Hofmann (2001) the word probabilities are weighted by tf-idf. Solving the log-likelihood function is carried out

by using the tempered EM-algorithm (Algorithm 3.4), analogously to the single models.

As a result, we obtain for each latent class z the vectors $(P(w|z))_{1\leq w\leq M}$ for the textual as well as $(P(c|z))_{1\leq c\leq N}$ for the link-based descriptions. These vectors are not created independently from each other but are folded with each other through the probability produced in the maximization-step.

Similar to the single-models, the results can serve for clustering procedures as well as for document representations from which a similarity measure can be calculated. By directly using the latent classes as clusters, those words and documents are extracted for each class that were generated with the highest probability from this class.

The clustering process is performed using the following scheme:

1. Generate a document set for the search query using HITS or ALA.
2. Perform extraction of links and text extraction, stop-word elimination and stemming.
3. Create a suffix tree in order to gain phrases.
4. Create cluster vectors with PLSI-PHITS with modified EM (Algorithm 3.4).
5. Extract cluster descriptions.
6. Assign documents to cluster descriptions.

Research of Hofmann (2000) leads to clusters which are on the one hand of a good quality but on the other hand mix subjects or seem to be randomly created. For this reason, the PLSI-PHITS model is implemented together with a cluster-description-first approach (analogously to the proceeding with the PLSI model). During the selection of the textual description, we only consider the textual information whereas with generating the latent classes textual as well as link-based information is considered.

As described in the PLSI model, using the suffix tree those phrases are selected among the extracted phrases that best fit to the latent classes. If a phrase is already used for a cluster then reject it. All documents that contain the phrase in their textual representation are assigned to a common cluster.

As a result, clusters are yielded with well-defined subjects, which have a qualitatively higher allocation compared to the results of Hofmann (2000).

Hierarchical Agglomerative Clustering (HAC)

The similarity measure gained based on PLSI-PHITS document-representation can be the basis for further clustering procedures. The following *hierarchical-agglomerative clustering* (HAC) procedure is characterized by its simple concept and its flexibility with regard to granularity. As a result it yields a hierarchical tree of clusters, which is also called dendrogram.

The procedure starts with a state where each document $d_i \in \{d_1, ..., d_N\}$ is contained in an own cluster $cl_j \in \{c_1, ..., c_K\}$, so initially $K = N$. It merges in each iteration-step those two clusters that fulfill the criteria of the target function best because of their pairwise similarities. The similarity between the documents within a cluster shall be as big as possible, and the similarity between two clusters shall be as small as possible.

The cosine coefficient is used as similarity measure. Combining the textual (sim_T) and the linkage (sim_L) similarity leads to a total similarity of

$$\text{sim}\,(d_i, d_j) = \alpha \cdot \text{sim}_T\,(d_i, d_j) + (1 - \alpha) \cdot \text{sim}_L\,(d_i, d_j)\,,$$

with given α ($0 \leq \alpha \leq 1$).

Besides document similarities, we also need a measure of similarities between the clusters. Here, we use the *average linkage* method that calculates the average similarity of all document pairs (d_r, d_s) of the clusters cl_i and cl_j with $d_r \in cl_i$ and $d_s \in cl_j$ as follows:

$$\text{sim}\,(cl_i, cl_j) = \frac{1}{n_i n_j} \sum_{\substack{d_r \in cl_i \\ d_s \in cl_j}} \text{sim}(d_r, d_s).$$

For the similarity $\text{sim}_\rho(d_r, d_s)$ of two documents d_r and d_s we use the cosine measure with $\rho = \text{T}$ for the text-based and with $\rho = \text{L}$ for the link-based calculation:

$$\begin{aligned} \text{sim}_\text{T}\,(d_r, d_s) &= \lambda \cdot \text{sim}_{cos}^{tfidf}\,(d_r, d_s) + (1 - \lambda) \cdot \text{sim}_{cos}^{PLSI}\,(d_r, d_s) \\ \text{sim}_\text{L}\,(d_r, d_s) &= \text{sim}_{cos}^{PHITS}\,(d_r, d_s) \end{aligned}$$

with $0 \leq \lambda \leq 1$.

The decision whether to merge two clusters or not depends on the self similarity of the resulting document group. The self-similarity of a group Φ of documents is

calculated as

$$\mathrm{sim}(\Phi) = \frac{2}{|\Phi|\,(|\Phi|-1)} \left(\alpha \cdot \sum_{d_i,d_j \in \Phi} \mathrm{sim_T}\left(d_i,d_j\right) + (1-\alpha) \cdot \sum_{d_i,d_j \in \Phi} \mathrm{sim_L}\left(d_i,d_j\right) \right)$$

with $0 \leq \alpha \leq 1$.

As the vector-space representation of the documents already exists in the L_2-normalized form, the similarities $\mathrm{sim}(d_i,d_j)$ correspond to the dot-product $\left\langle \vec{d}_i, \vec{d}_j \right\rangle$ of the vector representation $\vec{d}$ of the documents. Each group Φ of documents is represented by two group-profile vectors

$$g_\rho(\Phi) := \sum_{d \in \Phi} \vec{d}_\rho$$

with $\rho \in \{\mathrm{L},\mathrm{T}\}$.

The self-similarity measure proposed by Chakrabarti (2003)

$$\mathrm{sim_s}(\Phi) = \frac{\left\langle g_\rho(\Phi), g_\rho(\Phi) \right\rangle - |\Phi|}{|\Phi|\,(|\Phi|-1)}$$

is expanded for the combined consideration of link and text based information to

$$\mathrm{sim}(\Phi) = \frac{\alpha \left\langle g_T(\Phi), g_T(\Phi) \right\rangle + (1-\alpha) \left\langle g_L(\Phi), g_L(\Phi) \right\rangle - |\Phi|}{|\Phi|\,(|\Phi|-1)}.$$

Here, the calculation of the pairwise similarity-values can be reduced to the execution of dot-product calculations.

The number of clusters can be given by the number of iterations that are executed (here $\frac{N}{2}$). As an alternative stop-criterion, we can draw on the difference of the similarity values between two iterations.

After completion of the cluster generation, we look for descriptions for all clusters. Thus, we choose the phrase from the suffix array that exists in the most documents of the respective cluster. In case there are more candidates, we choose the phrase that consists of the most words.

The determination of the clusters is performed by the following scheme:

1. Generate a document set for the search query using HITS or ALA.
2. Perform extraction of links and text extraction, stop-word elimination and stemming.
3. Create a suffix tree in order to gain phrases.

Algorithm 3.5 Hierarchical agglomerative clustering (HAC) algorithm

```
procedure HAC(D)
  initialize set of clusters G with one-document
    clusters {d_i}: set G = {{d_i} | 1 ≤ i ≤ N)}
  while |G| > N/2 do {
    chose clusters Φ, Γ ∈
G with highest similarity sim(Φ, Γ)
    remove Φ and Γ from G
    set Δ = Φ ∪ Γ
    insert Δ into G
  }
```

4. Create vector representation of documents with PLSI-PHITS with modified EM (Algorithm 3.4).
5. Create a document-document similarity matrix.
6. Determine clusters with HAC (Algorithm 3.5).
7. Assign description to each cluster.
8. Verify cluster descriptions.

Ejection Chain Heuristic (EC and OEC)

Web documents can be regarded as vertices of a complete graph whose edges are weighted with pairwise similarities w_{ij}. A clustering of the documents can be achieved by partitioning the graph into subgraphs. Dorndorf and Pesch (1994) create with their *Ejection Chain* heuristic (EC) solutions for the problem known as *Clique Partitioning Problem*. In contrast to the probabilistic methods and the HAC, the number of clusters is not given and belongs to the model's output.

The target function can be formulated as a maximization of the sum of all edge weights w_{ij} within a cluster:

$$\max \sum_{1 \le k \le n} \sum_{\substack{i,j \in V_k \\ i \ne j}} w_{ij}$$

with the set V_k of vertices of the cluster k and n as number of the graph's vertices.

Introducing a variable

$$x_{ij} := \begin{cases} 1 & \exists k \in \{1,...,n\} \,|\, i \in V_k \wedge j \in V_k \\ 0 & otherwise \end{cases}$$

for every edge (i, j), the clique partitioning problem can be formalized as follows:

$$\max \sum_{1 \le i < j \le n} w_{ij} x_{ij}$$

s.t.

$$\begin{aligned} x_{ij} &\le 1 && (1 \le i < j \le n) \\ x_{ij} + x_{jk} - x_{ik} &\le 1 && (1 \le i < j < k \le n) \\ x_{ij} - x_{jk} + x_{ik} &\le 1 && (1 \le i < j < k \le n) \\ -x_{ij} + x_{jk} + x_{ik} &\le 1 && (1 \le i < j < k \le n) \\ |\{k \in \{1,...,n\} \,|\, i \in V_k \wedge j \in V_k\}| &= x_{ij} && (1 \le i < j \le n) \\ |\{k \in \{1,...,n\} \,|\, i \in V_k\}| &= 1 && (1 \le i \le n) \end{aligned}$$

If all edge weights are nonnegative or nonpositive, trivial solutions are produced consisting of all vertices in one cluster or each vertex in its own cluster. In order to obtain both, positive and negative edge weights, the mean value μ_{sim} of the similarity values is subtracted from each value:

$$w_{ij} = \mathrm{sim}(d_i, d_j) - \mu_{sim},$$

where the PLSI-PHITS cosine measure is used for the similarity measure $\mathrm{sim}(d_i, d_j)$.

The EC heuristic starts with the feasible solution X consisting of a random allocation of vertices to clusters. A feasible solution X' is called a neighbor of X if exactly one vertex is allocated to another cluster. By carrying out the right sequence of new allocations one gets from an arbitrary starting-point to the optimal solution of the clique-partitioning problem. In an iterative procedure, we choose that neighbor of the starting point that leads to a maximum improvement of the objective function value.

Let $v(u)$ be the actual cluster label of the vertex u and let $V_1, ..., V_k, ..., V_n$ be the clusters of the actual solution, so vertex u is located in cluster $V_{v(u)}$. For each vertex u we calculate as internal costs

$$I(u) = \sum_{\substack{i \in V_{v(u)} \\ i \ne u}} w_{ui}$$

the sum of all edge weights combining vertex u with the vertex in the same cluster.

We calculate for each vertex u as external costs

$$E(u,k) = \sum_{\substack{i \in V_k \\ i \neq u}} w_{ui}$$

the sum of all edge weights that combine u with the vertices of cluster k.

On re-arranging the vertex u to cluster k we yield as a gain

$$g(u,k) = E(u,k) - I(u).$$

Now, a sequence is created from n best new allocations by choosing the allocation with the highest gain, respectively. The highest gain is delivered by the allocation

$$g(u^*,k^*) = \max_{u \in V} \left\{ \max_{\substack{1 \leq k \leq n \\ k \neq v(u)}} \{g(u,k)\} \right\}.$$

A reassignment of a vertex u^* from the cluster $V_{v(u^*)}$ to cluster V_{k^*} leads to the following changes:

$$\begin{aligned} E(u, v(u^*)) &\leftarrow E(u, v(u^*)) - w_{u^* u} && \forall u \in V \setminus \{u^*\} \\ E(u, k^*) &\leftarrow E(u, k^*) + w_{u^* u} && \forall u \in V \setminus \{u^*\}. \end{aligned}$$

A problem of local search is that often only local optima will be reached that are far from the global optimum. By using tabu-markings, a further displacement of an already displaced vertex is avoided.

From the first n sub-sequences we choose the one whose $r = r^*$ allocations mostly improves the objective function value and the process is started over again. If there is no improvement, the original allocation is used (see Algorithm 3.6).

As Web documents can often not be thematically assigned to only one cluster, the EC heuristic is extended with the ability of creating overlapping clusters. This is reached by replacing the last constraint of the formal model by

$$|\{k \in \{1,...,n\} \,|\, i \in V_k\}| > 0 \quad (1 \leq i \leq n).$$

This constraint forces each vertex to be assigned to at least one cluster, which implies that each vertex is allowed to be in more than one cluster.

The *Overlapping EC* heuristic *(OEC)* introduces a duplication step to each iteration in which documents can be assigned to one or more clusters, additional to

Algorithm 3.6 Ejection Chain heuristic (EC) (Dorndorf and Pesch, 1994)

```
procedure OEC(D)
  initialization:
    randomly generate a feasible solution,
    compute all external and internal costs.
  repeat
    duplicate the current solution and
      mark all vertices as non-tabu.
    for i := 1 to n do { // find a seq. of n best moves)
      // move phase
      determine a non-tabu vertex that maximizes the
        gain g(u,k) for all u,k ∈
V and denominate this
        maximum g(u*,k*)
      perform the corresponding move and
        mark vertex u* tabu.
      update matrix E.
    }
    // improve the solution
    among all subsequences of the first 0 ≤ r ≤ n
      moves of the sequence of n moves let r*
      be a number of moves of a subsequence
      such that these first r* moves improve
      the solution at most.
    if r* > 0 then replace the current
      solution by its duplicate after
      performing these r* moves.
  until r* = 0;
```

their previous cluster. For monitoring the number of clusters, this step is carried out depending upon the fact that a defined minimum value of the resulting gain is exceeded.

The duplication of a vertex u in a new cluster k is performed as long as the following inequation is fulfilled:

$$E(u^*,k^*) = \max_{u \in V} \left\{ \max_{\substack{1 \le k \le n \\ k \neq v(u)}} \{E(u,k)\} \right\} > |k| \, threshold,$$

where $|k|$ is the number of vertices assigned to cluster k. The value *threshold* ($threshold \in \mathbb{R}$) sets a minimum value for the average similarity of a vertex u with all vertices u_k of the clusters k.

After completion of the clustering procedure, we need descriptions for the created clusters. They are chosen – like in the HAC procedure – from the amount of phrases extracted using the suffix-tree. The clustering procedure consists of following scheme:

1. Generate a document set for the search query using HITS or ALA.
2. Perform extraction of links and text extraction, stop-word elimination and stemming.
3. Create a suffix tree in order to gain phrases.
4. Create vector representation of documents with PLSI-PHITS with modified EM (Algorithm 3.4).
5. Create a document-document similarity matrix.
6. Reduce similarity values by the mean value.
7. Determine clusters with Overlapping EC.
8. Assign description to each cluster.
9. Verify cluster descriptions.

In the following, only the overlapping variant of the ejection chain algorithm is used. Thus, each mention of the ejection chain or EC method will refer to the overlapping EC (OEC) method.

3.5.5 Implementation

The described methods were implemented in the `C++` programming language. Figure 3.19 shows the main classes in a simplified way. The preprocessing and the

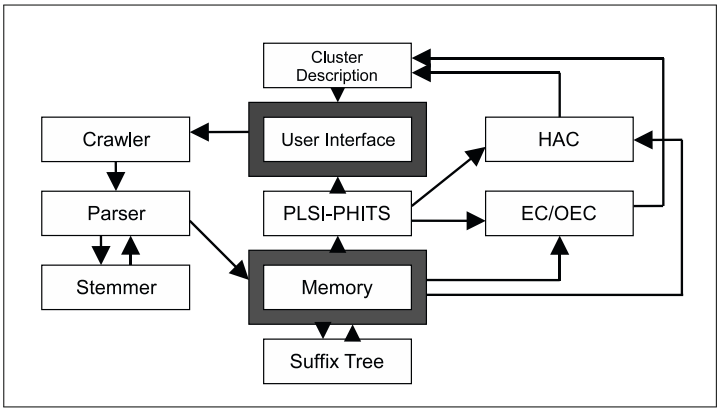


Figure 3.19: Framework of implemented classes

cluster algorithm are grouped around the two central classes: The *memory* class holds the data structures needed for the clustering process and the *user interface* class controls the execution of the different classes.

The user interface of the implemented solution (Figure 3.20) shows the choice of different methods and the (empty) result boxes. It contains three areas:

1. Creation of a data set (upper area)
2. Clustering procedure with result boxes (central area)
3. Status bar (lower area)

The creation of a data set is performed after the entry of a query term. An existing data set can also be loaded or stored by the system. The *crawler* class requests search engine result pages and transmits the HTML code to the *parser* class. The parser creates sets of words and eliminates stop words. The stemmer writes the words that are reduced to their stem as well as the original (unstemmed) words into the memory. Phrases are stored in the *suffix tree* from words that were not reduced.

After choosing a clustering method in the central area of the main window, the corresponding classes are invoked.

The PLSI-PHITS representation is created by the PLSI class and forwards their results to the document clustering methods HAC and EC/OEC. The HAC and the EC/OEC can also directly access the memory. Both methods transmit their results

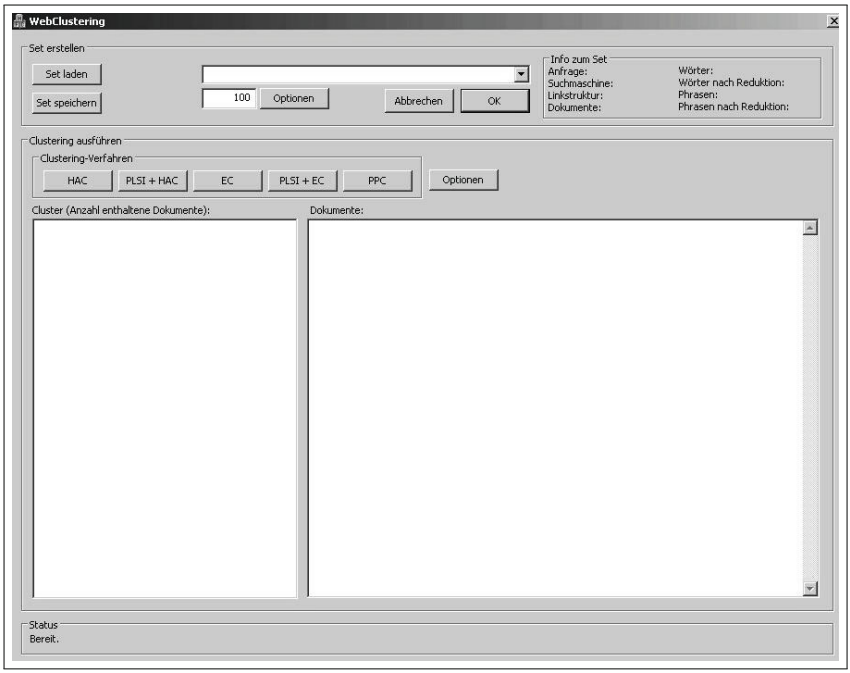


Figure 3.20: User interface

to the *cluster description* class that creates appropriate descriptions. As the PPC approach generates cluster descriptions itself, there are no further classes needed. The results of the PLSI-PHITS representation directly provide clusters and their descriptions.

The left result box of the central area contains the determined cluster descriptions with the number of documents contained in the cluster. The right box holds the document URLs included in the cluster selected by the users.

The status box informs the user of the current working status of the program.

3.5.6 Experimental Results

In order to evaluate the results we need a benchmark for their quality. This benchmark can be the distance to an "ideal" solution. An "ideal solution" consists of finding the "right" clusters and of the "right" allocation of the documents to these clusters. In the context of a document categorization, we know the categories in

Dataset	A			B
Query	"jaguar"			"statistical learning"
Variant	I	II	III	I
Textual Information	yes	yes	yes	yes
Links (Generation Method)	no (–)	yes (ALA)	yes (HITS)	no (–)
Documents	100	107	93	100
Words	812	588	486	664
Words after Reduction	176	95	106	169
Phrases	3853	2171	1655	3164
Phrases after Reduction	223	108	143	367

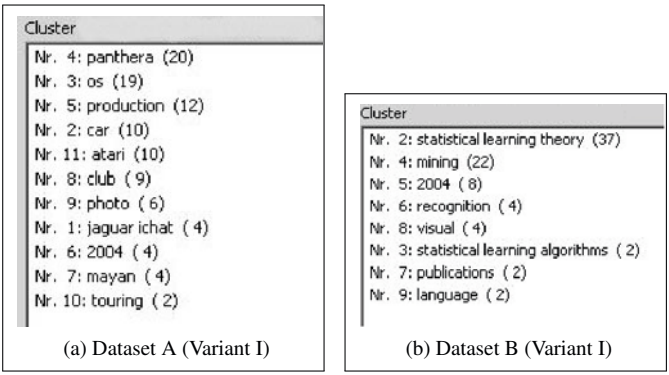
Table 3.3: Test datasets with variants – content type and item counts

advance and it is sufficient to determine the quality of the procedures from the quality of allocation. Furthermore, categorization procedures imply that an allocation of the documents is at least possible using the given information. Existing Web directories can also be used as a comparison. In the directories, however, general categories are installed that are not specific to a certain kind of research.

In the following, the results of the procedure presented here are qualitatively examined. Therefore, two test-data sets are created from results of the search engine GOOGLE.

In *Dataset A*, we choose as an example for an unspecified inquiry the search word "Jaguar" from which we expect search results from very different areas. Besides the wild animal the word "Jaguar" stands e.g. for an automobile brand or a version of the system software MAC OS. The search results are generated in three ways: *Variant I* exclusively uses textual information, *Variant II* uses textual information as well as the link information determined with ALA, and *Variant III* uses textual information and the link information determined with the HITS algorithm. The search query of *Dataset B* contains a more specific issue and consists of the term "statistical learning". For the *Dataset B* we exclusively use *Variant I*.

Table 3.3 gives an overview of the contents and generation methods of the datasets and their variants. It shows the number of documents in the sets, the



(a) Dataset A (Variant I) (b) Dataset B (Variant I)

Figure 3.21: Results for HAC after 90 iterations – cluster labels and numbers of documents contained

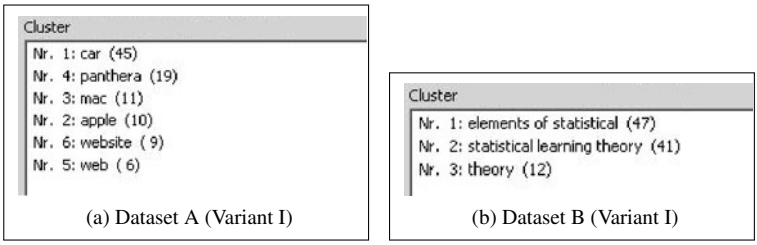
number of words before and after stop-word reduction, and the number of phrases before and after the reduction.

Text Based Clustering

First, an evaluation of the clustering procedures based on Variant I of the datasets A and B containing only textual information is carried out.

During the HAC procedure, the number of clusters is determined through the number of iteration steps, which can be arbitrarily chosen. After 50 iteration steps 94 documents have been assigned to 34 clusters. A tendency towards descriptions that only consist of one word shows up. It is sometimes difficult to deduce the topic from the description without regarding the cluster content, especially in the case of one-word descriptions. In the result set, there is for example a cluster with the description "production". It does neither become clear which production is meant nor to which product it refers.

In approximately 14 % (Dataset A) and 13 % (Dataset B) of the cases, documents are assigned to wrong clusters, i.e. clusters whose remaining documents do not possess a visible coherence to them. An increase of the number of iteration steps to 90 leads to different observations on the datasets. The number of clusters decreases, whereas the number of documents per cluster grows. The unspecific query of Dataset A results in larger clusters of the core topics like the animal ("panthera"), the operating system ("OS"), the car, or the video game console ("atari"). Figure 3.21a shows the results after 90 iterations. The less specific query of Dataset B leads to a stronger concentrations of the documents in the first



(a) Dataset A (Variant I) (b) Dataset B (Variant I)

Figure 3.22: Results for EC-PLSI – cluster labels and numbers of documents contained

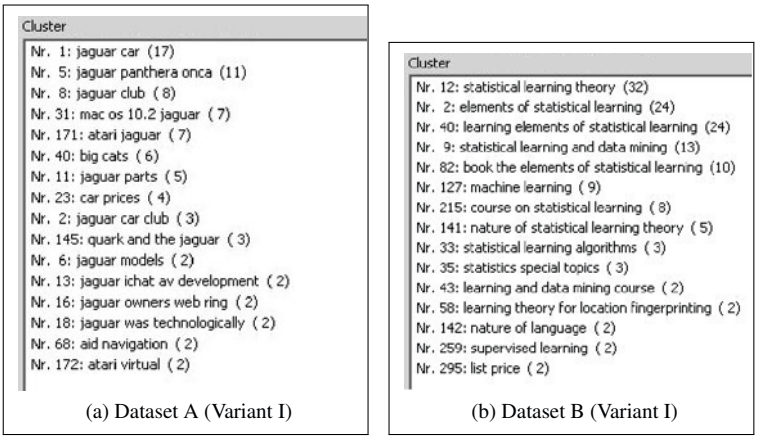
clusters, because the topics of the query are not that distinctive (compare Figure 3.21b).

However, the allocation accuracy decreases with increase of the number of iteration steps. E.g., some automobile related URLs are allocated to the cluster "os" even though there is not any relation to the operating system visible. The same tendency can be observed in all clusters of both datasets. In Dataset A, 38 % of the documents are allocated to wrong clusters compared to 35 % of wrong allocations in Dataset B.

The EC heuristic combined with the PLSI yields only a small number of clusters. One reason for this is the (arbitrary) positioning of the algebraic-sign-limit by subtraction of the average value from the similarity value. It determines the number of clusters that could be changed by a different weighting. The clusters well present the groups of subjects (see Figure 3.22). The precision of the cluster contents show a slightly minor accuracy compared to the other methods described. The clusters of both datasets contain over 40 % of documents from different topics.

In contrast to HAC and EC, the PPC procedure yields longer and more precise cluster descriptions (compare Figure 3.23). The cluster descriptions contain better information about the cluster content. The results for Dataset A contain the core topics already described. Particularly in Dataset B, the PPC procedure yields with a similar number of clusters considerably more significant descriptions.

A reason for the better performance of PPC is the use of the cluster-description-first procedure. It prevents the allocation of documents to clusters to which they have no or only a slight reference. Therefore, there are no allocations from documents to clusters with diverging topics. The superior performance of the cluster-description-first method results from the often poor quality of base data. In many datasets extracted for a certain query, there are documents without a content relationship with the context of the query. This phenomenon basically occurs because of the expansion of the document set with e.g. HITS or ALA. The extraction of



(a) Dataset A (Variant I) (b) Dataset B (Variant I)

Figure 3.23: Results for PPC – cluster labels and numbers of documents contained

	PLSI-PHITS	Clustering	Total
HAC (50 iterations)	5.4 s	1.8 s	7.2 s
EC	5.3 s	1.2 s	6.5 s
PPC	5.5 s	–	5.5 s

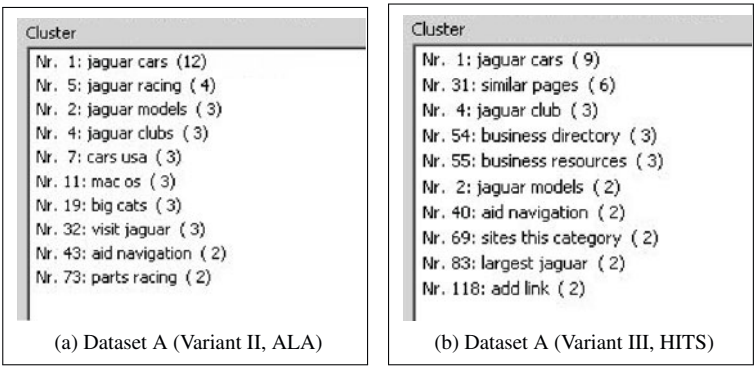
Table 3.4: Processing times of clustering methods

cluster descriptions by PLSI-PHITS excludes these documents because they belong to different topics and do not possess textual similarities with relevant documents. In contrast, the document clustering methods (HAC, EC and OEC) assign at least one cluster to every document. This negatively influences the cluster quality.

Processing Times

Table 3.4 shows the processing times for the analyzed methods. For the document clustering methods HAC and EC, the times needed for processing the document representation (PLSI-PHITS) and the clustering algorithm are distinguished. Since the PPC approach only consists of one step, it does not consume any time for a supplemental clustering algorithm.

All times are measured using the `clock` operation based on the system clock of the computer (PC with AMD Athlon XP 2500+ CPU and 512 MB RAM).



(a) Dataset A (Variant II, ALA) (b) Dataset A (Variant III, HITS)

Figure 3.24: Results for PPC including link structure ($\alpha = 0.5$)

The PLSI-PHITS algorithm runs up to three times longer than the clustering algorithm. All methods need more than 5.5 seconds. This duration is basically caused by the complex PLSI-PHITS procedure (compare Table 3.4). The clustering part of the EC algorithm only needs 30 % less processing time than the HAC algorithm. Regarding the total processing times, the PPC is the fastest algorithm.

Influence of Link Structure

Based on the PPC procedure, which yielded the best clusters, the influence of link-based information on the result is examined. For this purpose, link-information is included in two variants of Dataset A. In Variant II of Dataset A the aroundlink algorithm (ALA) is used to gather links. The HITS algorithm is used to gain the links included in Variant III of Dataset A. The textual information and the link structure are equally weighted with $\alpha = 0.5$ and thus $(1-\alpha) = 0.5$, respectively, in the similarity matrix (compare Section 3.5).

A comparison of the results of Variant II including the link structure gained with ALA (Figure 3.24a) with the results of Variant I (Figure 3.23a) yields the following observations. Fewer clusters are determined, and the clusters are smaller although a similar number of documents is included. Two new clusters called “jaguar racing” and “parts racing” are added. All other cluster descriptions of Variant I were already contained in Variant II in identical or comparable form. A few other topics got lost compared to the text-only variant. This loss can be explained by the different formation of the datasets. The influence of the link structure on the results of PPC is minor. A change of the link proportion to $(1-\alpha) = 0$ yields seven of ten clusters that are identical.

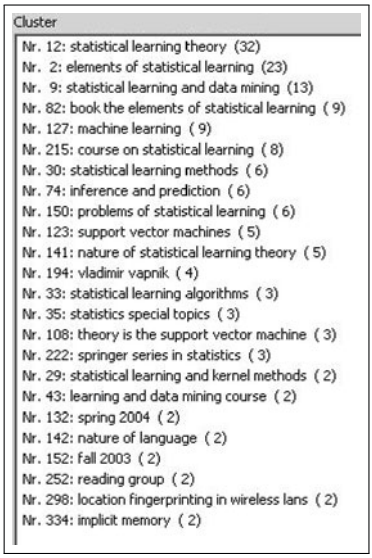


Figure 3.25: Results for PPC with 40 latency classes (Dataset B, Variant I)

A comparison of the results of the HITS based Variant III (Figure 3.24b) with the text-based Variant I (Figure 3.23a) yields similar observations. In this case also, fewer and smaller clusters are created. Some of the clusters (like "business directory" or "add link") do not have any coherence with the query. This can be explained by an additional topic drift that occurs with the HITS algorithm (compare Subsections 2.4.2 and 2.6.2). Here, URLs that are not relevant with regard to the query were incorporated in the datasets.

Influence of Latency Classes

In the PLSI-PHITS model, the number of latency classes used can be modified. Increasing the number of latency classes from $K = 20$ to $K = 40$ doubles the running time of the whole clustering process. The results for Dataset B Variant I with $K = 40$ shown in Figure 3.25 are compared with Figure 3.23b ($K = 20$). There are some identical clusters (like "statistical learning theory"). The additionally occurring clusters bear a high resemblance with the results of 20 classes. The high number of clusters leads to very specific cluster descriptions and overlapping clusters like "theory is the support vector machine" and "support vector machines".

Figure 3.26: Results of VIVISIMO

Comparison with Vivisimo

If we compare the PPC procedure to the clustering engine VIVISIMO similar clusters are extracted (see Figure 3.26). Because of the bigger data basis of VIVISIMO, a higher number of clusters is created. Despite the smaller data basis, the PPC procedure is able to identify a big part of the subjects and to extract further subjects. Thus, PPC partly produces more accurate cluster descriptions, but VIVISIMO returns results in a shorter timeframe.

Summary and Outlook

The procedures presented here are suited to offer an additional value to the user by adequately summarizing search results into clusters. In comparison, the PLSI-PHITS clustering performs better and faster than the hierarchical-agglomerative clustering and the ejection-chain procedure. Advantages in the (perceived) quality of clusters arise from the cluster-description-first procedure. The form of link-information generation with the ALA and the HITS procedure seems to deteriorate the results because of the topic drift and therefore leads to a minor performance of

these procedures that use the link-information. The simultaneous usage of textual descriptions and link-based methods currently creates some inaccuracy. A better use of the link-information can be the subject of further work so that their use can better contribute to the quality of results.

4 Search Engine Optimization

Companies, organizations or persons that are offering websites in the Internet are interested in people regarding their content. Depending on the goals of the specific website, they usually want a high number of the "right" people to visit their website. The goals can be to spread information or opinions for a certain topic or make the visitor perform some transactions. Ideally, the goals of the website owner correspond to one or some of the users' needs described in Subsection 1.1.1.

Since most people use search engines and the corresponding user numbers are increasing (compare Figure 1.1, Table 1.1 and Chapter 1) and these users concentrate their attention on the first results on the result pages (compare Table 1.4 and Subsection 1.3.5), it has become important for website owners to appear on high ranks in order to be found by their addressees. The process of improving the volume and the quality of visitors on a website coming via search engines is called *search engine optimization (SEO)*. Most SEO activities aim for high positions on the result pages because click rates are usually higher on top positions.

After showing the goals, possibilities and limits of search engine optimization, the effects will be tested and evaluated in a practical project implementing SEO methods in a Web design process.

4.1 Objectives

The website owner pursues certain goals by publishing his content in the World Wide Web. Independent of the fact if he or she runs a shop, distributes opinions or establishes a Web community, the owner is interested in reaching the target audience. These efforts of the website owner can be subsumed under his marketing activities.

Search engine optimization can be one component of the owner's marketing activities. The well known *marketing mix* model established by McCarthy (1975) consists of four areas (the so-called four "P"s): Product, Price, Place and Promotion. SEO activities fall under the promotion area (also known as communication policy). The promotion consists of individual and mass communication, brand management and corporate identity (compare standard marketing literature like Kotler and Keller, 2008). A good search engine placement of a website does not

Measure	Short description
1. Traditional Media Measures	
Recall	Remembrance of an advertisement
Recognition	Recognition of an advertisement
2. Contact Measures	
Hits	Number of requested files
Page Impressions	Number of views of a webpage
Ad Impressions	Number of views of an advertisement
Visits	Number of visits of a website
Unique Users / Visitors	Number of unique visitors on a website
Banner Reach	Number of visitors having viewed the banner
Banner Frequency	Number of views per visitor
3. Interactivity Measures	
AdClick	Number of clicks on a banner
AdClick Point-in-Time	Point in time of a click on a banner
4. Result Oriented Measures	
Transactions	Number of transactions performed
Turnover	Realized turnover
Contribution Margin	Realized contribution margin
New customer registration	Number of newly registered customers

Table 4.1: Advertising effectiveness measures (Skiera and Spann, 2000)

only serve the mass communication component. SEO can also contribute to the success of brand management and the communication of a corporate identity.

If applying SEO, the SEO goals have to be adapted to those of the regarded website's owner. For persons or companies offering their products or services on the Web, the typical economic goals of communication policy are to increase transaction volume, turnover, market share, or profit. They may also want to decrease costs. The website owner's communication policy often pursues psychographic goals, too: The owner may want to improve the awareness of a product, service or company. He aims at increasing the publicity of his brand or to distribute knowledge about it. A positive attitude towards a brand and a brand image shall be established. The preference for the offered good or a purchase intention shall be awoken. These psychographic goals can be easily adapted to non-profit organizations, too.

SEO helps to reach these goals by aiming to increase the quality and quantity of visitors on a website. Skiera and Spann (2000) have assembled effectiveness measures for advertisements and have structured them into four groups: *traditional media measures, contact measures, interactivity measures,* and *result oriented measures* (compare Table 4.1). While the first group refers to classical advertisements and the third group only aims at banner advertisements, the measures contained in the second and fourth group (contact measures and result oriented measures) are suitable for monitoring the ability of SEO and thus are considered in more detail.

The first contact measure, the number of *hits*, counts all files requested by a browser and delivered by the server. As an HTML page may consist of multiple elements (and files), multiple hits can be counted after a user has viewed only one single page. The *page impression* measure overcomes this problem by counting all hits on the same page's elements as one page impression. For monitoring banner advertisements on webpages, the *ad impressions* measure is relevant. The *visits* measure counts multiple, consecutive page impressions of the same user on pages of the same website as one visit. The number of *unique users* or *visitors* is measured by assigning a unique identification attribute (like a cookie) for a longer time period in order to identify revisits. On this basis, the number of visitors having contact with a banner at all *(banner reach)* and their average contact frequency over a time period *(banner frequency)* can be measured. Thus, for measuring the achievement of website goals with SEO means, the numbers of page impressions and visitors are used.

While contact measures can be used to monitor both, economic and psychographic goals, the *result oriented measures* refer to purely economic indicators. These can be e.g. the number of *transactions*, the *turnover*, the *contribution margin* or the number of *newly registered customers*.

Figure 4.1 illustrates the necessary steps towards the achievement of economic goals using the example of an online shop. The shop is assumed to aim at maximizing its turnover and its profit (*economic goals,* measured by result oriented measures of Table 4.1). This can be reached by increasing the number and volume of transactions on the store's website performed by existing and new customers.

The achievement of economic goals can be positively influenced by increasing the number of page impressions and visitors on the website *(website goals)*. The achievement of website goals can be monitored by the contact measures of Table 4.1.

Subsection 4.1.1 describes methods capable of increasing the number and quality of visitors. The number of visitors on a website can be augmented by reaching upper positions on search-engine result-pages (SERP). Table 1.4 and Subsection

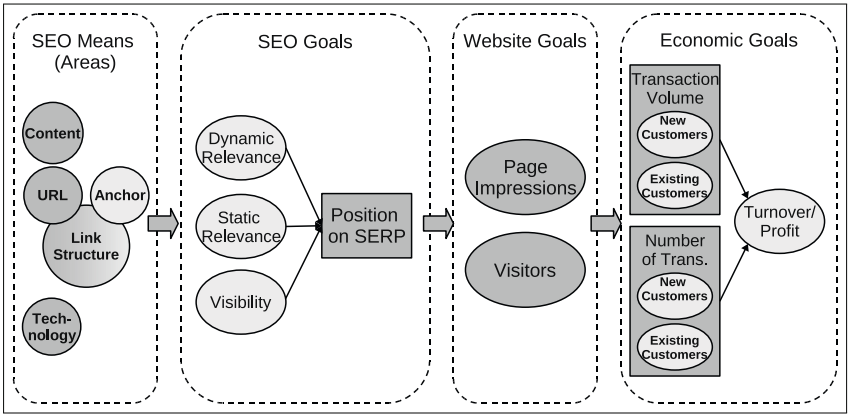


Figure 4.1: SEO impact chain

1.3.5 show that upper positions are mostly viewed and consequently mostly clicked upon. Thus, reaching high positions on SERPs and influencing the dynamic and static relevancy as well as the visibility of webpages are *SEO goals*. The functioning of these influence mechanisms is described in Subsection 4.1.2.

The different *SEO means* to reach the SEO goals will be described in Section 4.2. Using the right *technology* assures a webpage's visibility for search engines. The *link structure* including suitable *URL* names and *anchor* texts positively influences the static relevance of a webpage. Finally, the right *content* assures that the webpage will be found by the right users. For this purpose, different keyword selection models will be introduced in Section 4.3.

4.1.1 Website Visitors

One of the goals of a website owner is to maximize the number of visitors on the website. The positions on which the webpages of a site appear on a search-engine result-page influence the number of users visiting this website. The following model aims at explaining this impact.

We assume that there are H webpages, represented by the documents d_h ($h = 1,...,H$). For the sake of a simple model, websites will not be considered separately of webpages. Thus, webpages are the subject of analysis. The conclusions for webpages can be transformed to websites as they can be regarded as a set of webpages.

Let the number of different query terms that can be entered in the search engine be specified by $I \in \mathbb{N}_0$. There are $u_i \in \mathbb{N}_0$ users (searchers) searching for a query term $q_i \in Q$ $(i = 1,...,I)$, where $Q \subseteq strings$ is a set of possible query terms. The number of users u_i is constant for each i $(i = 1,...,I)$ and is given exogenously. Let us define a function

$$\text{position} : \{1,\ldots,H\} \times \{1,\ldots,I\} \to \mathbb{N}$$

that assigns a position on the SERP to every document h for each query i. A possible definition of this function is presented in Subsection 4.1.2. For this model, the position j_{hi} of a document h on the SERP for a query i is determined by the position function as

$$j_{hi} = \text{position}\,(d_h, q_i)\,.$$

There are J positions available: $j_{hi} \in \{1,\ldots,J\}$. A value of $j_{hi} = 1$ denominates the top position on the SERP.

The total number of visitors v_h on a document d_h consists of the sum of visitors $v_h^s \in \mathbb{N}_0$ coming from search engines and visitors $v_h^e \in \mathbb{N}_0$ coming from other sources: $v_h = v_h^e + v_h^s$ $(h = 1,...,H)$. v_h^e is assumed to be constant because it cannot be influenced in the framework of this model. v_h^s consists of the sum of all visitors reaching the document h via different queries i:

$$v_h^s = \sum_{i=1}^{I} v_{hi} \qquad (h = 1,\ldots,H),$$

where $v_{hi} \in \mathbb{N}_0$ is determined by a function

$$\text{visitors} : \{1,\ldots,J\} \times \{1,\ldots,I\} \to \mathbb{N}_0.$$

The observations described in Subsection 1.3.5 show that documents on higher positions are viewed more often than those on lower positions. Thus, the number of visitors

$$v_{hi} = \text{visitors}\,(j_{hi}, u_i)$$

depends on the position j_{hi} of document h for a search term i and the number of users u_i that search for the term i. The function is assumed to have the following property for a given number of users u_i. For a given i and for each pair of documents (d_h, d_m) with $h, m \in \{1,\ldots,H\}$ and $h \neq m$ the conclusion

$$j_{hi} < j_{mi} \Rightarrow v_{hi} \geq v_{mi}$$

is true. This means for each query i that if document h holds a higher position (thus a lower value of j_{hi}) as document m then the number of visitors of document h is not lower than the one of document m.

The underlying assumptions allow the following conclusions. For maximizing the total number of visitors reaching document h by applying

$$v_h = v_h^e + \sum_{i=1}^{I} v_{hi} \rightarrow \max,$$

the values v_{hi} must be as high as possible for every i. From the relationships of positions with visitor numbers follows

$$\begin{aligned} & v_h && \rightarrow \max \\ \Leftrightarrow \quad & v_{hi} && \rightarrow \max \ (\forall i) \\ \Leftrightarrow \quad & j_{hi} && \rightarrow \min \ (\forall i)\,. \end{aligned}$$

The highest number of visitors is expected on a document h if the position value for as many as possible search terms is as low as possible.

Assuming that a part of the visitors on a document generate sales or transactions with a constant contribution margin, the profit can be maximized by minimizing the position values, too. This relation will be used in the keyword selection models in Section 4.3. The next subsection explains how the position can be influenced.

4.1.2 Result Page Position

The position of a document on the SERP depends on multiple factors. The main influence factors can be grouped into three categories.

The *visibility* of a document determines whether a search engine is able to read a document at all or not. Invisibility is often caused by the technology used to construct the website. Most search engines can only interpret information that is available in textual form. Words that are e.g. for layout reasons transformed into bitmap pictures are usually not recognized as long as these words are not added as alternative text for the picture. The same is true for most textual content included in script languages. A document can either be visible or not visible for a search engine. A state in between is not possible.

For each visible document that is included in a search engine's index, a *static relevance* value can be calculated. It measures a kind of importance of the doc-

Dynamic Relevance \ Static Relevance		Low	High
Low		Page does not appear at all or appears at the bottom of the result list.	Page may only be found accidentally.
High	for irrelevant keywords	Page appears at the bottom of the result list.	Wrong users will find page (possibly perceived as SPAM).
	for relevant keywords	Page appears at the bottom of the result list.	The right users will find page.

Figure 4.2: Effects of static and dynamic relevance on visible pages

ument among all other documents on the Web. Methods to calculate relevance values are presented in Subsection 2.4.1.

The *dynamic relevance* measures the relevance of a document with regard to a certain query. It estimates how good a document may fit to the query.

Effects of static and dynamic relevance

The combination of static and dynamic relevance values decides on the order documents appear on the SERP. Figure 4.2 shows the impact of the relevance measures on the webpage position.

If the static relevance of a page is low related to other pages the page may only appear at the bottom of the result list or it will not appear at all. In practice, the latter two alternatives do not make a big difference because the user's main focus is on the upper positions, anyway. Moreover, a high dynamic relevance for certain keywords is not able to bring webpages to top positions if the static relevancy is low.

If the static relevance of a document is high but the dynamic relevance for a query is low, the page cannot be found by these queries. It remains on lower position and may only be found accidentally.

If the static relevancy of a document is high and the dynamic relevance is only high for keywords that do not fit to the contents of the page (for irrelevant keywords) users may find the webpage although they are searching for a different topic. If the positioning of the document complicates the search for relevant doc-

uments, i.e. if the user searching for a keyword finds only irrelevant documents, they can be perceived as spam.

Only if both, the static and the dynamic relevance for the right keywords contain high values, the document will be found by the right users.

Visitor Position Model

The following model explains the interrelation between visibility, static and dynamic relevance, and the position on the SERP. The position function introduced in Subsection 4.1.1 assigns a position to each document h for a query i. The position of a document depends on the relevance values the search engine calculates for documents in relation to all other documents fitting to the query. Assuming the relevance values are assigned by a function $R : \{1,\ldots,H\} \times \{1,\ldots,I\} \to \mathbb{R}$, the position of a document h depends on the relevance of the document h itself in relation to all other documents $m \in \{1,\ldots,H\} \setminus h$:

$$\begin{aligned}
&\text{position}\,(d_h,q_i)\\
&= f\,(R\,(d_1,q_i),\ldots,R\,(d_{h-1},q_i),R\,(d_h,q_i),R\,(d_{h+1},q_i),\ldots,R\,(d_H,q_i))\\
&= f\,(R\,(d_h,q_i),\{R\,(d_m,q_i)\,|m \in \{1,\ldots,H\} \setminus h\})\,.
\end{aligned}$$

The website owner is able to influence the relevance value of his own documents. Assuming that he is not interested in improving the relevance value of other documents and that he is not able to downgrade other documents, the relevance values for all documents $m \in \{1,\ldots,H\} \setminus h$ are external parameters in respect to the minimization of the position value. Consequently, a reasonable influence is only possible on the document h. Thus, maximizing $R\,(d_h,q_i)$ is sufficient in order to minimize $\text{position}\,(d_h,q_i)$ for a query i:

$$\begin{aligned}
&\text{position}\,(d_h,q_i) \to \min\\
\Leftrightarrow &R\,(d_h,q_i) \to \max\,.
\end{aligned}$$

It follows from the relation between visitors and position derived in Subsection 4.1.1 that the maximization of relevance values leads to the maximization of the visitor numbers on document h:

$$\begin{aligned}
&\forall i : R\,(d_h,q_i) \to \max \Leftrightarrow \forall \text{i} : \text{position}\,(d_h,q_i) \to \min\\
\Rightarrow &\forall i : v_{h,i} \to \max\\
\Rightarrow &v_h \to \max\,.
\end{aligned}$$

Even though the exact parameters for the calculation of relevance performed by public available search engines is kept secret, knowing the main calculation elements is sufficient for influencing the ranking position. A relevance function can be written (analogously to the consideration in Schaale et al., 2003) as a product of three factors

$$R(d_h, q_i) = R^v(d_h) \cdot R^s(d_h) \cdot R^d(d_h, q_i) \ (i = 1, ..., I; h = 1, ..., H).$$

The binary variable $R^v(d_h) \in \{0; 1\}$ indicates the visibility status of document d_h. Regular HTML documents that can be found by search engines are usually visible, thus receive a value of $R^v = 1$. If a document d_h cannot be found or cannot be read by search engines, the visibility gets a value of $R^v(d_h) := 0$. If a document h is not visible, the search engine consequently assigns a relevance value of $R(d_h, q_i) = 0$ for any query i to the document.

Depending on the quality and quantity of incoming links, each document receives a static relevance value $R^s(d_h) \in \mathbb{R}^+$ that does not dependent on the query. Higher static relevance increases the total relevance if the visibility and dynamic relevance contain values greater than zero.

The better a document h fits to a query i, the higher its dynamic relevance $R^d(d_h, q_i) \in \mathbb{R}^+$ will be. As with the static relevance, R^d can only positively affect the total relevance if the visibility and the static relevance consist of positive non-zero values, too.

Thus, all three factors need to be regarded in order to reach high total relevance values. The function will not be filled with numerical parameters as the exact algorithms of search engines are not published and are subject of continuous modifications. For the purpose of improving the total relevance, it suffices to analyze the direction of causality. Possibilities and means to influence the factors are topic of the next section.

The model can be extended by regarding the quality of visitors additionally to the visitor quantity. The quality can be measured according to the economic goals enumerated, e.g. by the expected transaction volume. This variant will be regarded in separate keyword selection models.

4.2 Possibilities and Means

This section presents different possibilities and means to increase the relevance values and thus improve the position and number of visitors of a website. This does not include those positions on SERPs that are reached by paying money

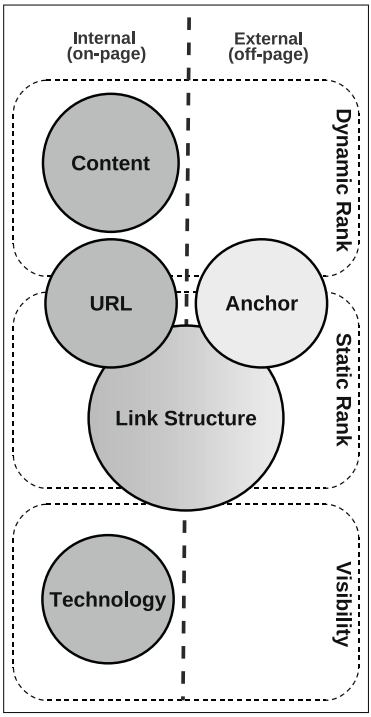


Figure 4.3: Approaches for increasing rank and visibility by location, effect and type

to the search engine owner *(paid advertisement)*. Here, only the mechanisms to reach better positions by improving relevance and visibility *(organic listing)* are regarded.

The approaches can be classified by different dimensions (compare Figure 4.3): by location, effect and type. SEO activities can be performed on two locations: on the webpage itself *(on-page* or *internal)* and on other webpages *(off-page* or *external).* They can also be classified by their effect into the three factors of the position model: dynamic rank, static rank and visibility. The circles in Figure 4.3 represent different types of SEO activities. Their positions on the figure indicate their assignments to their activity location and relevance factor.

There is a multitude of practical advices in literature that describe how to increase result-page positions (see e.g. Alby and Karzaunikat, 2007; Schwarz, 2008; Erlhofer, 2007; Glöggler, 2003; Chung and Klündler, 2007). This section takes a

selection of promising methods and categorizes them under the factors of the position model prior to integrating them in a Web design process presented in Section 4.5.

4.2.1 Visibility

The visibility of a webpage is a necessary precondition for the inclusion into a search engine's index. Even though a regular HTML page should be readable for search engines, there may occur circumstances under which the webpage becomes invisible for crawlers. An analysis of a Web document's visibility helps its owner to correct or to avoid errors in this regard.

Firstly, it must be assured that the URL of the webpage is known or at least can be known by search engines. One possibility is to announce the URL at one or multiple search engines that provide forms to fill out on their website. Usually, it is sufficient to place the URL on another webpage that is already in the index of a search engine. In this case, the crawler of the search engine can find the URL at its next crawling process and adds it to his list of pages to be visited. When changing the URL name of a page, the problem of publishing this URL must be considered, too.

Secondly, the indexing of the document should not be forbidden by entries in the file `robots.txt` (compare Section 2.2). Even though some search engines do not respect the *robots exclusion protocol*[1] and read excluded pages anyway, such exclusion may hinder the webpage from being crawled. Whether a document is included in a search engine's index or not can be easily checked.

After these two preconditions for making the document findable have been fulfilled, some measures have to be taken in order to make the document furthermore readable. Since a search engine only stores the textual part of a Web document, the amount of stored information can be checked by using a text browser like LYNX or the text mode of the OPERA browser.[2] These tools visualize the image a search engine receives of the document. The case that some of the information which the website presents to a human audience is missing in this image can be caused by several, mostly technical reasons (as indicated in Figure 4.3).

Search engines will only add a limited amount of data per webpage to their indexes. All information that exceeds this amount will be cut off. Consequently, the webpage cannot be found using parts of the missing content in the search query. A possible solution is to divide large pages into multiple smaller pages.

[1] see `www.robotstxt.org`
[2] `www.opera.com`; `lynx.isc.org`

Another problem occurs if a webpage consists of multiple frames. Although the human viewer of the page perceives only one webpage consisting of different elements, search engines regard each element as a single document. Thus, if keywords of the same query are distributed over different elements, the search engine will not be able to implicate the relation between the elements and consequently will not return multiple frames of the same webpage as one result.

In case a search engine returns one sub-frame of a webpage because it has found the keywords in this sub-frame, only this frame will be shown on the result page. The search-engine user will only see this sub-frame after clicking on the result, unless the website owner has implemented an automatism that reloads the remaining sub-frames of the webpage.

A webpage has to consist of correct HTML code. Otherwise a search engine may possibly not read the content as intended. The webpage can be validated manually or using HTML validator tools in order to assure that correct HTML code is used.

Also the usage of dynamically generated pages may cause some problems for search engines. In this case, some webpage content is stored in databases and will only be generated and transformed into HTML at runtime. A webpage may be adapted to a user request that is expressed in parameters or session IDs. The problems search engines may have with such URL structures can be addressed by rewritten URL names. Some content management systems (CMS) offer this service. The content of dynamically generated pages and of those pages built by script languages like JAVASCRIPT can be additionally provided using plain HTML that can be read by search engines.

Finally, a reliable Web server is an important precondition for visible Web documents. Even if the Web server fails only at the time the crawler visits one of the hosted documents, this can lead to the fact that these documents will not be listed in the search index.

4.2.2 Static Relevance

The measures for increasing the static relevance of a webpage are mainly related to the link structure. This comprises the (internal, on-page) hyperlink structure of a website and the (external, off-page) hyperlink connections with other websites, as well as the structure of URL names (on-page) and anchor texts (off-page) on other pages (compare Figure 4.3). The effect of these measures is restricted by the limits described in Section 4.4.

On-page Measures

The analysis of the internal link structure needs to regard the single documents as well as the whole website. All webpages should be reachable from the homepage of the website. A flat hierarchy of documents helps the crawler to reach the documents on a short way. As documents further down the structure tend to be valued as less important, essential documents should be connected as directly as possible to the homepage. This implies also a short URL name, which is also an attribute of an important document. The number of outgoing links of a webpage should not be too high, as it dilutes the distribution of PageRank and may be seen as a characteristic of spam.

The choice of the URL names can support the perception of a flat hierarchy if a long structure with multiple slash separators is avoided in favor of short names. A content management system can perform the renaming of URLs if necessary. In order to have sustainable effects of these measures, the URL names should not change more often than necessary over time.

Off-page Measures

The most effective but also most difficult method to increase the static relevance of a webpage is to add external hyperlinks pointing to it. According to the PageRank definition (compare Section 2.4.1), a webpage receives a high rank value if other pages possessing a high rank point to it. For this reason, hyperlinks on highly ranked pages are treasured by many website owners and accordingly hard to get.

The website owner needs to seek for opportunities to place valuable links. Links can also be placed in return for payment (paid links) or in return to a placed link on the own page (link exchange). Both methods are not appreciated by search engines as they undermine their algorithms' output quality. As with other SEO methods, this can lead to a total exclusion of this link from the PageRank calculation (compare limits in Section 4.4). The text related to the placed link (anchor text) itself does not contribute to the static rank, but it may help to increase the dynamic relevance of the linked page.

4.2.3 Dynamic Relevance

The dynamic relevance is calculated in order to evaluate the suitability of a Web document for a certain query. Thus, the query term or a part of it must either occur in the Web document or the URL name or stand in any other relation with the document, e.g. through an anchor text. The dynamic relevance mainly depends on the textual content of webpages, anchors and URLs.

The constitution of appropriate content depends on the website goals. If the website owner wants to increase only the number of users regardless of their quality, he or she may want to reach high positions on the result pages independently from the query used. Here, a strategy is to fill the webpage with keywords many users are searching for. Another strategy is to use keywords in the document's content that preferably few other Web documents contain. Those website owners that want a certain clientele to visit their sites can adapt the page content to the keywords this clientele is supposed to search for. In this work, the focus lies on the latter group of website owners. A model for selecting the right keywords is presented in Section 4.3.

The content of a webpage needs to be relevant to the searcher. If it is unique, it is easier to reach higher result-page positions for related queries. As there are webpages stuffed with certain keywords for the only reason of attaining high result-page positions, search engines try to separate such artificial content from natural content. Webpages with artificial content are threatened to be excluded from the search index. Thus, the text structure of Web documents should be as natural as possible with the right keywords occurring in a natural quantity. The word density distribution should not differ from other natural texts. In order to adapt to the variety of search terms, keywords should be used with different flection. Company specific or technical terms should be translated into everyday language. Also, using synonyms helps the potential searcher to find the webpage using other words. Adding a glossary of the terms used is an appropriate tool for comprising different explanations for keywords.

The HTML structure elements (e.g. "`<h1>...</h1>`" and "`<h2>...</h2>`" for header level 1 and 2 or "`<title>...</title>`" for the document title) can be used to emphasize certain keywords in the document. Here, a modest usage of the elements is recommendable. Table 4.2 shows metatags the website owner can use to describe the document content and give additional information about its author, language, keywords and more. These metatags are also used by search engines. Some display the `description` tag's content on their result page.

HTML elements can also be used to describe the content of pictures that otherwise cannot come into the searchable index. If the alternative text of images is filled with appropriate keywords, the webpage can also be found in this way.

4.3 Keyword Selection

The selection of the right keywords is crucial for the search engine optimization process. Regarding the SEO impact chain in Figure 4.1, the keyword selection

```
<meta http-equiv="Content-Type" content="text/html>
<meta http-equiv="Content-Language" content="de">
<meta name="audience" content="alle,all">
<meta name="Author" content="XXXXXXXXXX">
<meta name="copyright" content="XXXXXX">
<meta name="Description" content="XXXXXX">
<meta name="GENERATOR" content="XXXXXXXXX">
<meta name="KeyWords" content="XXXXX">
<meta name="language" content="de">
<meta name="page-topic" content="XXXXXX">
<meta name="page-type" content="XXXXXX">
<meta name="publisher" content="XXXXX">
<meta name="revisit-after" content="10 days">
<meta name="robots" content="index,follow">
```

Table 4.2: Sample Metatags

influences the dynamic relevance of a Web document and thus its total relevance. In SEO literature, the search terms a webpage is optimized for are often referred to as *keywords*. In order to consider also queries consisting of more than one word, the more exact notion *search term* is used in the model description. The models are named using the more common notion *keyword*, anyway.

The position of a webpage on the search-engine result-page depends on the search term entered. Increasing the positions for the right keywords positively influences the quality and the quantity of visitors, which are the basis for generating profit out of the website.

This section presents different models that explain the interrelation between the influenceable parameters and the business goal. Each model aims at maximizing the profit by selling products. A fix amount of customers u_i ($u_i \in \mathbb{N}_0$) comes via search engines to the website after looking for certain search terms q_i ($i \in \{1,...,I\}$). It is assumed that the website owner is able to improve the position of his webpages by putting in a certain effort. This effort is expressed by an amount of money spent. The cost of reaching the top position for search term q_i amounts to $C_i \in \mathbb{R}^+$.

The percent sign is used as an operator that divides the preceding percentage by 100. The function $\% : \mathbb{R} \to \mathbb{R}$ is defined by $p\% = \frac{p}{100}$.

4.3.1 Keyword Selection Position Model (KSPM)

In this subsection, a *Keyword Selection Position Model (KSPM)* is introduced. It determines how much effort a website owner should put into the optimization for which keyword or search term. By doing so, the model is able to select those search terms that can be profitably used for SEO.

Product, Document and Search Term Variables

For creating a simple model, we assume that one webpage relates to exactly one product. This webpage can be reached by the search term q_i so that i relates to a product, a Web document and a search term at the same time. The profit margin of each product i is given by $m_i^p \in \mathbb{R}^+$. With a given percentage of visitors $v_i^b \in \mathbb{R}^+$, $0 \leq v_i^b \leq 1$, that buy product i after searching for search term q_i, one can calculate an expected profit margin of search term q_i per visit on document i as $m_i = m_i^p v_i^b$.

Position Variables

The positions on the search-engine result-page are numbered with $j = 1,...,J$ where $j = 1$ denominates the top position. Users click on a listed webpage with a probability k_j ($0 \leq k_j \leq 1$) that depends on the position j. A smaller position number leads to a higher click rate so that $j < q \Rightarrow k_j \geq k_q$ is true for each $j = 1,...,J$ and each $q = j+1,...,J$. It is assumed that the cost for reaching a position j can be calculated as percentage c_j ($0 \leq c_j \leq 1$) of the cost for reaching the top position C_i. Here, a smaller position number is related with higher relative cost, so that $j < q \Rightarrow c_j \geq c_q$ is true for $j = 1,...,J$ and $q = j+1,...,J$.

Model

The expected profit for a product i is calculated from the expected revenues less the costs spent for the placement of the product on the result page. The revenue is generated by an expected number of visitors $v_{ij} = k_j u_i$ on the webpage i if it is listed on position j. The profit margin of each of these visitors is expected to be m_i, which leads to an expected revenue of $k_j u_i m_i$. The cost of reaching position j for document i can be calculated as $c_j C_i$.

After introducing the binary decision variable

$$x_{ij} := \begin{cases} 1 & \text{if document } i \text{ appears on position } j \\ 0 & \text{otherwise,} \end{cases}$$

j	c_j	k_j	$\frac{c_j - c_{j+1}}{k_j - k_{j+1}}$
1	100%	15%	20
2	60%	13%	10
3	30%	10%	$\frac{c_j}{k_j} = 3$

Table 4.3: Sample position parameters

the model maximizing the profit can be formulated as

$$\max \sum_{i=1}^{I} \sum_{j=1}^{J} (k_j u_i m_i - c_j C_i) x_{ij}$$

s.t.

$$\sum_{j=1}^{J} x_{ij} \leq 1 \qquad (i = 1, \ldots, I)$$

$$x_{ij} \in \{0;1\} \qquad (i = 1, \ldots, I; j = 1, \ldots, J).$$

The constraints assure that each search term i is assigned to at most one each result page position j.

The best solution for each term i can be calculated determining

$$j_i^* = j \,\big|\, (k_j u_i m_i - c_j C_i) \rightarrow \max .$$

The effort put in the improvement of the search engine position of product i is profitable if $k_{j_i^*} u_i m_i \geq c_{j_i^*} C_i$ is true. With other words, one can spend up to an amount of $k_{j_i^*} u_i m_i$ for placing product i on position j_i^*.

Example

The model is explained using an example with $I = 5$ products, search terms and webpages and $J = 3$ positions. Search engine users viewing a result page click with a probability of $k_1 = 0.15$ on the first position, and with $k_2 = 0.13$ and $k_3 = 0.1$ on the following positions. Table 4.3 gives an overview over the values used. The cost for reaching the top position of a keyword is defined by $c_1 \cdot C_i = 100\% \cdot C_i = C_i$. The following positions cost $c_2 = 60\%$ and $c_3 = 30\%$ of the top positions cost C_i.

Five products are represented by five webpages (see Table 4.4). Each webpage is reached by exactly one search term which is the same as the product name. The parameters are explained using the first product ($i = 1$) called “mobile phone”.

i	1	2	3	4	5
product webpage search term	mobile phone	organizer	music player	navigation system	camera
u_i	800	1'200	600	400	50
v_i^b	5%	8.5%	3%	2%	6%
m_i^p	320	200	400	100	300
m_i	16	17	12	2	18
C_i	200	1'500	800	200	400
$u_i m_i$	12'800	20'400	7'200	800	900
$\frac{u_i m_i}{C_i}$	64	13.6	9	4	2.25

Table 4.4: Sample product parameters

i	1	2	3	4	5
$j = 1$	* 1'720	1'560	280	–80	–265
$j = 2$	1'544	* 1'752	456	–16	–123
$j = 3$	1'220	1'590	* 480	* 20	* –30

Table 4.5: Solution of example – expected profit of product per position

$u_1 = 800$ users are searching for the term "mobile phone" during the regarded time period. If a user clicks on a search result and visits the webpage for mobile phone, he or she buys this product with a probability of $v_1^b = 5\%$. Considering the profit margin of $m_1^p = 320$ achieved at the sale of a mobile phone, this leads to an expected profit margin per user of

$$m_1 = m_1^p v_1^b = 320 \cdot 5\% = 16.$$

The cost of reaching the top position for the search term "mobile phone" is $C_1 = 200$. The second position on the result page costs $c_2 \cdot C_1 = 60\% \cdot 200 = 120$.

Calculating the expected profit of the second position for search term "mobile phone" results to an expected revenue of

$$k_2 u_1 m_1 = 13\% \cdot 800 \cdot 16 = 1'664.$$

This revenue is achieved spending $c_2C_1 = 120$ and thus, a profit of

$$k_2u_1m_1 - c_2C_1 = 1'544$$

is achieved. Table 4.5 shows the results for all products and positions. In this example, choosing the maximum value per column leads to an optimal solution (marked by an asterisk in the table). The table shows as result that it is worth to spend money to reach the first position for mobile phone, as this position generates the highest profit. In this solution, the second position is optimal for an organizer. As the profit is negative for all positions, it is not worth to spend any money at all for reaching one of the top three positions for the product $i = 5$ camera.

Influence of Position Parameters

The influence of the position parameters c_j and k_j can be analyzed as follows. Subtracting the condition for j

$$k_ju_im_i \geq c_jC_i$$

from the condition for $j+1$

$$k_{j+1}u_im_i \geq c_{j+1}C_i$$

leads to

$$\left(k_j - k_{j+1}\right)u_im_i \geq \left(c_j - c_{j+1}\right)C_i.$$

Because of $k_j > k_{j+1}$ and $C_i > 0$, a transformation to

$$\frac{u_im_i}{C_i} \geq \frac{c_j - c_{j+1}}{k_j - k_{j+1}}$$

is feasible.

The inequality for product $i = 2$ is true for position $j = 2$:

$$\frac{u_2m_2}{C_2} = 13.6 \geq \frac{c_2 - c_3}{k_2 - k_3} = 10.$$

It is not true for position $j = 1$:

$$\frac{u_2m_2}{C_2} = 13.6 \not\geq \frac{c_1 - c_2}{k_1 - k_2} = 20.$$

Thus, the optimal solution for $i = 2$ is $j_2^* = 2$. This means that the product $i = 2$ "organizer" should be placed on the result-page position 2 for a maximum profit.

Conclusion
Using this model, the website owner can determine for which search terms he should invest in SEO activities in order to maximize his profit. With the same calculation, the maximum profitable cost spent for SEO activities for reaching a certain position can be determined.

A problem of the model arises because of the assumption that each webpage is only reached by one search term. In practice, there are multiple different search terms leading to the same webpage. The next subsection introduces a solution for this problem.

4.3.2 Enhanced Keyword Selection Position Model (EKSPM)

The *Enhanced Keyword Selection Position Model (EKSPM)* aims at overcoming the problems arising out of the unity of document and search terms. Thus, it allows multiple search terms per single webpage.

Product and Document Variables
For this reason, a new index $h \in \{1,...,H\}$ is introduced for H products represented by one Web document each. The profit margin $m_h^p \in \mathbb{R}^+$ accordingly refers to the product h.

Search Term Variables
Each document h can be reached by different search terms $i \in I_h$. The set of search terms I_h is a subset of all possible search terms ($I_h \subseteq \{1,...,I\}$ for $h = 1,...,H$). The number of users u_i searching for term i and the cost C_i for reaching the top position for term i are defined as in 4.3.1.

Search Term and Product Variables
A new parameter v_{hi}^b is introduced for modeling the percentage of visitors that buy product h if they have searched for term i ($0 \leq v_{hi}^b \leq 1$). This is caused by the fact that in practice, different buying frequencies are observed on the same webpage depending on the search term the visitor has used. By setting $v_{hi}^b := 0$ for every $i \notin I_h$ and $h = 1,...,H$, the mathematical formulation does not have to explicitly consider the single subsets I_h but can regard all search terms.

Consequently, the expected profit margin for a product h also depends on the search term the users come from. It is calculated as $m_{hi} = m_h^p \cdot v_{hi}^b$.

Position Variables
The position variables $j = 1,...,J$, k_j and c_j are defined as in Subsection 4.3.1.

Model

Introducing the decision variable

$$x_{hij} := \begin{cases} 1 & \text{if document } h \text{ appears on position } j \text{ for query term } i \\ 0 & \text{otherwise} \end{cases},$$

the EKSPM can be formulated as

$$\max \sum_{h=1}^{H} \sum_{i=1}^{I} \sum_{j=1}^{J} \left(k_j u_i m_h^p v_{hi}^b - c_j C_i \right) x_{hij}$$

$$s.t.$$

$$\sum_{j=1}^{J} x_{hij} \leq 1 \qquad (h = 1,...,H; i = 1,\ldots,I)$$

$$\sum_{h=1}^{H} x_{hij} \leq 1 \qquad (i = 1,\ldots,I; j = 1,\ldots,J)$$

$$x_{hij} \in \{0;1\} \qquad (h = 1,...,H; i = 1,\ldots,I; j = 1,\ldots,J).$$

As in the KSPM, the profit is maximized. The first constraint assures that each document h is assigned to at most one position j on the search-engine result-page for search term i. The second constraint assures that each position j of a result page for search term i is only filled with one document h.

The introduction of separate parameters for search terms and documents leads to more difficulties when trying to find a feasible solution of the problem. A solution can be to split the model into different models for each search term and solve these models separately for each search term. By doing so, the relation between Web document and search term is kept, but their interdependencies are neglected.

Input Values and Problems

Some of the input values are more difficult to obtain than others. The profit margin m_h^p for each product h is known by the shop owner. The number of users u_i searching for a term i is published by search engines (e.g. GOOGLE ADWORDS) and can be downloaded on their websites. The buyer percentage of visitors v_{hi}^b can be estimated from the analysis of historical values using the website's log files in combination with the sales system. The cost C_i for reaching the top position for a

term i is more difficult to determine. As estimation, one can use the cost for search engines advertisements like GOOGLE ADWORDS. These values do not directly refer to the cost of reaching the top position, but they indicate the relation between the prices of different search terms.

Problems occur during the estimation of the position related parameters. The relative click rates k_j of the positions j are usually not published by search engines. One possibility is the application of general values coming out of user studies. This implies the assumption that the k_j is constant over different keywords which is not necessarily true. Similar problems occur for the relative costs c_j. These costs are hard to obtain as they are not publicly available. Also the relative costs may vary from search term to search term and should not be assumed to be constant.

4.3.3 Keyword Selection Top Position Model (KSTPM)

The *Keyword Selection Top Position Model (KSTPM)* overcomes problems arising from the insufficient availability of the data needed for the EKSPM. By focusing only on top positions of documents for specific search terms, different values for k_j and c_j are not needed anymore. This model is of great practical relevance for the case that the EKSPM is not applicable because k_j and c_j cannot be (properly) determined. The KSTPM answers the question which search term to focus on. The answer to this question is crucial for practical SEO applications.

Thus, a constant rate $k_1$of users clicking on the position of a result page is assumed. This leads to a number of users searching for term i of $u_i' = u_i \cdot k_1$. The remaining variables are defined as in the EKSPM.

Model

Introducing the decision variable

$$x_{hi} := \begin{cases} 1 & \text{if document } h \text{ appears on top position for query term } i \\ 0 & \text{otherwise} \end{cases},$$

the profit maximization of the KSTPM can be formulated as

$$\max \sum_{h=1}^{H} \sum_{i=1}^{I} \left(u_i' m_h^p v_{hi}^b - C_i \right) x_{hi}$$

s.t.

$$\sum_{h=1}^{H} x_{hi} \leq 1 \qquad (i = 1, \ldots, I)$$

$$x_{hi} \in \{0; 1\} \qquad (h = 1, ..., H; i = 1, \ldots, I).$$

The constraints assure that for each keyword i only one document h can reach the top position.

This model is relatively easy to solve as follows. Profitable search terms i can be determined with

$$i_h^* = i \left| \left(u_i' m_h^p v_{hi}^b - C_i \right) \rightarrow \max \right.$$

for each product $h \in \{1, \ldots, H\}$. The effort put into the improvement of the search engine position of search term i for product h is profitable if $u_i' m_h^p v_{hi}^b \geq C_i$ is true. Thus, the website owner can spend an amount of $u_i' m_h^p v_{hi}^b$ for the search term i in order to promote product h.

As this model is easy to solve and all input data is available, the KSTPM is well suitable for practical application. With its help, appropriate search terms can be chosen as basis for processing the SEO task list, which is presented in Section 4.5.

4.4 Limits

The effects of the possibilities and means described for improving the SERP position of a webpage are limited by several factors. Limits are caused by other website owners, by user behavior, by cost or capacity restrictions, or by search engines.

Usually, there are different actors in the Web canvassing visitors with similar interests. If there are other website owners competing for good SERP positions in respect to the same keywords, the own success is limited by the success of the competitors. The more interesting a keyword is for the website owners, the more the top positions are fought over. Game theoretic approaches can help to explain these interrelations.

If users find websites with a poor connection to the query on top positions and this is caused by extensive SEO measures of these websites, they may perceive these websites as spam, and the site owners may lose reputation. This is espe-

cially the case if the documents the users expect to find are displaced onto worse positions.

In case there are users that click on heavily promoted pages, the website owner will more fulfill his website goal to increase page impressions or visitors. But if these users are not at all interested in the product or service offered, the website owner will not approach his business goals like sales volume. On the contrary, more visitors on a page consume bandwidth and sometimes the related costs are charged by the Internet provider. Moreover, a huge amount of visitors may exceed the Web server capacities. Under these circumstances, SEO measures can even counteract the business goals.

As search engines provide a service to their users, one of their primary interests is to satisfy them. This comprises the fulfillment of the users' needs to find what they are searching for. If some website owners overuse the possibilities and means of SEO with the consequence of a worse SERP quality, this counteracts the search engines' efforts to deliver as relevant results as possible.

4.4.1 Search Engine Spam

Users perceive webpages as unwanted content or spam, if these pages appear on the SERP and are not related to the query or only very distantly related. This perception of the same document may vary from user to user and thus, its evaluation is a subjective decision. One of the challenges search engines are faced is to identify spam and remove it from their indexes (compare Henzinger et al., 2002). The website owner using SEO methods needs to be careful that his measures are not classified as spamming by the search engines, because an exclusion from the index of an important search engine can significantly influence visitor numbers. The exclusion also forms a kind of punishment and thus may last longer than the original SEO measure taken is present. Besides the exclusion of a single webpage, the whole domain, the IP address or the IP group can be excluded.

In order to avoid confusions over which SEO measures are forbidden, search engines establish rules for webmasters. The quality guidelines of GOOGLE (Google, 2009) advise webmasters to avoid tricks intended to improve search engine rankings. The specific guidelines recommend not using the following techniques that are explained in more detail in the next subsection:

- Hidden text or hidden links.
- Cloaking or sneaky redirects.
- Loading pages with irrelevant keywords.

- Creating multiple pages, sub-domains, or domains with substantially duplicate content.
- "Doorway" pages created just for search engines, or other "cookie cutter" approaches such as affiliate programs with little or no original content.

Website owners try to find new ways that are not (yet) forbidden or cannot be easily detected by search engines to improve the positions of their pages. This leads to an enduring competition between search engines and optimizers.

Search engines continuously verify their index for abnormalities. In order to identify content that is artificially generated, e.g. word densities are measured (Ntoulas et al., 2006). Duplicate content is identified by comparison of different webpages. The methods presented in Subsection 2.4.3 can be used to identify artificially placed hyperlinks.

The following subsection will give an overview of typical and often used spamming techniques.

4.4.2 Spamming Methods

When optimizing webpages, one has to be careful not to cross the fine line between good optimization and spam. A high competition between webmasters can occur during the process of webpage optimization for commercially interesting keywords. Sometimes, aggressive methods can result in short term success, but on the long run, search engines will suppress the results of these methods. Anyway, in high competitive areas like pharmacy products or pornography, website owners go beyond the limits defined by search engines for the purpose of getting ahead of other competitors even by a narrow margin. This subsection gives an overview over aggressive methods that are occasionally perceived as spamming.

Text spamming

Some webpages are created for the only purpose of attracting the searchers of one single keyword. These pages are stuffed with this keyword, but try not to attract attention of search engines. Different locations in the HTML code of the page come into consideration for placing the keywords. The keywords are repeatedly inserted in the text body of the document. In order to not confuse the human reader of the document, the keywords are written in color of the background or beyond the visible areas of the document. Thus, the keywords are only visible for the search engine and not for the visitor. Keywords are also repeatedly placed in the title or other meta-tags of the documents, in HTML comments or in alternative attributes of pictures. Sometimes multiple title tags are used. The latter methods can be

classified as spam if the keywords are repeatedly used or if they do not have any further relation with the document's content. Only using these tags for moderate placement of keywords usually is not a problem for search engines. Another way to attract users is to copy content from other sources like DMOZ or WIKIPEDIA. Search engines try to identify this duplicate content. The challenge is to find out which page is the original one and which one is the copy.

Link spamming
Analogous methods are used for placing hyperlinks. Some spammers create large networks of webpages linked among each other for the purpose of increasing the PageRank (so called *link farms*). In doing so, they create multiple different URLs for pages with the same content *(shadow domains)*. Hyperlinks are placed in hidden form on own webpages, e.g. using one-pixel pictures. Publicly available guest books, bulletin boards or blogs are filled with comments containing hyperlinks. For this reasons, some search engines do not take into account entries of guest books while calculating ranking values.

Page spamming
There are also whole pages with content created for the only purpose of attracting search engine users. So called *doorway pages* are optimized for single keywords. Multiple of these pages are placed on the Web, sometimes only slightly varying from each other in order not to raise suspicion of being duplicate pages. For this reason, they present other content to the search engines as to the users *(cloaking)*. This is achieved using technical methods by e.g. forwarding the user with the refresh meta-tag, with JAVASCRIPT, CGI or with dynamically generated pages (PHP, ASP).

Webmasters using these spamming methods face the risk of being punished by the search engines. Thus, these methods should not be used by serious website owners. When using search engine optimization methods, one needs to thoroughly consider these limits in order to achieve ones goals.

For further reading, see Castillo et al. (2008) who give an overview of literature about web spam.

4.5 Implementation

A list of SEO tasks was elaborated for the implementation of possibilities and means of optimization described in Section 4.2. This list can be used as a checklist for existing websites and is presented in the following subsection. Thereafter, a website creation process is modeled and the SEO tasks are assigned to the process

phases for the purpose of considering SEO measures during the creation of new websites.

4.5.1 Tasks

After defining SEO tasks based on the possibilities and means, the single tasks were compiled into the list shown in the Figure 4.4. The list contains tasks to be performed for improving the quality of a website with respect to high positions in search-engine result-pages. The tasks are grouped into ten areas explained in the following. Different measures are identified additionally to tasks belonging to the five SEO approaches shown in Figure 4.3. The last three columns of the SEO task list are explained and used in the next subsection.

The identification and *definition of keywords* is separated from the content measures because it has to be performed independently from implementation of keywords. The keyword selection models presented in Subsection 4.3 can support in completing the tasks of this area.

The *content* area comprises different tasks to adapt the content to the needs of increasing the dynamic relevance for the identified keywords. The *technology* area mainly supports the visibility of the webpages. The tasks of the *link structure* area aim for increasing the static relevance.

AREA	TASK	Process Phase	Repetition	Report
Keyword Definition				
	Define keywords from searcher's point of view	P3 Detailed Concept	-	-
	Define keywords per website			
	Define keywords per webpage			
Content				
	Insert keywords per webpage	P4 Implementation	In case of intense content modification	-
	Fill descriptions			
	Fill page topic			
	Fill page title			
	Additional content			
	Create glossary			
	Add different languages			
	Set geo tags			
	Add alternative tags for pictures			
Technology				
	Meaningful URLs	P4 Implementation	-	-
	Validate HTML code			
	Use keywords in file names			
	Ensure visibility of links			
	Set base tag			
	Check tables			
	Check document size			
	Optimize pdf documents			
	Eliminate frames			
	Eliminate flash			
	Duplicate JavaScript links in HTML			
	Reduce parameters and special characters in dynamic URLs			
	Reduce session IDs			
	Eliminate cookie obligation			
	Check Web server speed at peak load			
	Create robots.txt			
Link Structure				
	Verify own links	P6 Operation	Quarterly	Report
	Verify inlinks of competitors	P3 Detailed Concept	-	
	Remove dead links	P6 Operation	Quarterly	Report
	Create 404 error catching page	P4 Implementation	-	
URL (internal)				
	Register site map at search engines	P5 Launch	-	-
	Maintain domain names	P3 Detailed Concept	-	
	Ensure that most links point on most important pages	P4 Implementation	-	
Anchor / PageRank (external)				
	Request DMOZ entry	P6 Operation	-	-
	Create cheap links in directories, wikis, organizations, universities	P6 Operation	Ongoing	
	Make links accessible from main domain	P4 Implementation	-	
	Copy competitors' links	P6 Operation	-	
	Create links from other own sites	P6 Operation	Quarterly	
	Formulate text anchors	P6 Operation	Quarterly	
	Put (varying) keywords in anchor texts	P6 Operation	Quarterly	
	Create organic link structure from different class-C blocks	P6 Operation	Quarterly	
Search Engine Result Control				
	Keep records of PageRank	P6 Operation	Monthly	Report
	Verify that domain is in index of search engines		Quarterly	-
	Verify if all pages are indexed		Quarterly	Report
	Verify descriptive texts on result pages		Quarterly	-
	Eliminate duplicate content		Quarterly	-
User				
	Look for possibilities to put domain in the Web	P6 Operation	Quarterly	-
Analysis				
	Analyze access logs	P6 Operation	Quarterly	Report
Other				
	Apply for trademark protection if necessary	P3 Detailed Concept	-	-

Figure 4.4: SEO tasks per area assigned to process phases and reports

The *URL* area assures appropriate domain names and internal link structures. In addition, the site map of the website is registered at search engines to ensure that all pages of the site are at least known by the search engines.

The *anchor and PageRank* area contains tasks for increasing the dynamic as well as the static relevance. For the static relevance, links are added to different locations in the Web. From the different locations, the hyperlinks in the OPEN DIRECTORY PROJECT DMOZ usually have a considerable impact on the relevance values. An organic link structure means that the links should not give the impression to be artifically created.

In the *result control* area, the inclusion status of webpages in search engines is monitored. The PageRank values are recorded to be able to react on changes by intensifying the application of some measures or identifying and correcting possible errors. If duplicate content is detected in search engines, it will be eliminated.

The *user* area does only indirectly belong to the SEO tasks. The user task aims in the first instance for possibilities to put the own domain or URL name on the Web where target user may click it. This increases the visitor number, and as a side effect makes search engines finding the URL in the right context.

The *analysis* area contains a task to evaluate and monitor the results the websites returns. The access logs are analyzed using custom made or publicly available tracking tools (e.g. ETRACKER or GOOGLE ANALYTICS[3]).

The last area *(other)* contains a task that becomes necessary if a new domain is registered. A trademark protection of the own domain name should be applied for in order to be sure to keep the domain under own control after having increased the PageRank.

4.5.2 Website Creation Process

A more convenient way to implement the SEO tasks is to incorporate them already when creating a new website. For this reason, a Website creation process is modeled. The process is designed from the point of view of a company that constructs websites for external clients. The goal of modeling the process is to identify the process steps where the SEO tasks have to be performed.

Process Phases

The process consists of six phases numbered from P1 to P6 (compare Table 4.6). It starts with the *acquisition* of a new project (P1) and *marketing* activities (P2). The *implementation* (P4) is performed based on a *detail concept* created in phase P3.

[3] `www.etracker.com; www.google.com/analytics`

No.	Process Phase
P1	Acquisition
P2	Marketing
P3	Detail Concept
P4	Implementation
P5	Launch
P6	Operation

Table 4.6: Phases of website creation process

After the *launch* (P5) of the website, an *operation* phase (P6) follows. Each task of the SEO task list is assigned to one of the process phases in the third column of Figure 4.4.

Process Flow

Figure 4.5 illustrates the process flow of the website creation. In the first two columns, it differentiates the process steps performed by the Web design company together with the client from the steps performed internally by the Web design company by itself. The content creating tasks and the SEO tasks are assigned to each phase in the third and fourth column of the process flow.

In the first phase *(P1 Acquisition)*, the Web design company communicates their online marketing knowledge to potential customers. During this phase, there are not any SEO tasks performed yet.

The second phase *(P2 Marketing)* starts with the collection of ideas, which results in a discussion concept. This basic concept is discussed together with the client. It is agreed upon the amount of necessary client co-operation. The basic concept already contains a definition of the amount and structure of the website to be developed. Based on the concept, an offer is submitted to the client. If an agreement is achieved and the client approves the offer, the next phase can start.

An internal project meeting elaborates a detail concept during the third phase *(P3 Detail Concept)*. The detail concept contains the functionality, the menu structure, the SEO measures and the keywords the website should be found with using search engines. For this purpose, the content of the website must be defined, that means either received from the client or generated by the Web design company. In this phase, the first SEO tasks are conducted. The domain name is registered and a webpage announcing the forthcoming website is created. If necessary, an application for trademark protection should be filed. Opportunities for placing hyperlinks

pointing to the announcing webpage are identified and some preliminary links are placed in order to start building a static relevancy.

After the agreement upon the detail concept with the client, the fourth phase *(P4 Implementation)* can start. The implementation tasks are performed internally. Based on the detail concept, a graphical site layout is created and implemented in HTML. After the website is programmed, an internal quality and usability check is performed. During this phase, the website content is integrated in either a database or a content management system (CMS). Furthermore, different implementational SEO tasks are performed. Besides the tasks of the content and technology area, the internal link structure is created and assured to be consistent. An error catching page is created.

In the fifth phase *(P5 Launch)*, the implemented website is presented to the client. The website is launched after the client agreement and approval. In this phase, a current site map is registered at search engines.

The launch is followed by the last phase *(P6 Operation)*. Change requests are collected during the operation of the website. A reporting and success measurement is set up for internal and client information purposes. The content is continuously maintained and different SEO measures are taken. The hyperlinks on the website are verified and removed if necessary. External hyperlinks pointing to the own site with appropriate anchor texts are created in the Web. The results returned by search engines are controlled for important keywords. The fourth column of the SEO task list shows a suggested repetition period for the ongoing tasks. The data used as a basis for reporting is marked in the fifth column of the task list. If necessary, larger change requests of the website are implemented in a second release of the website.

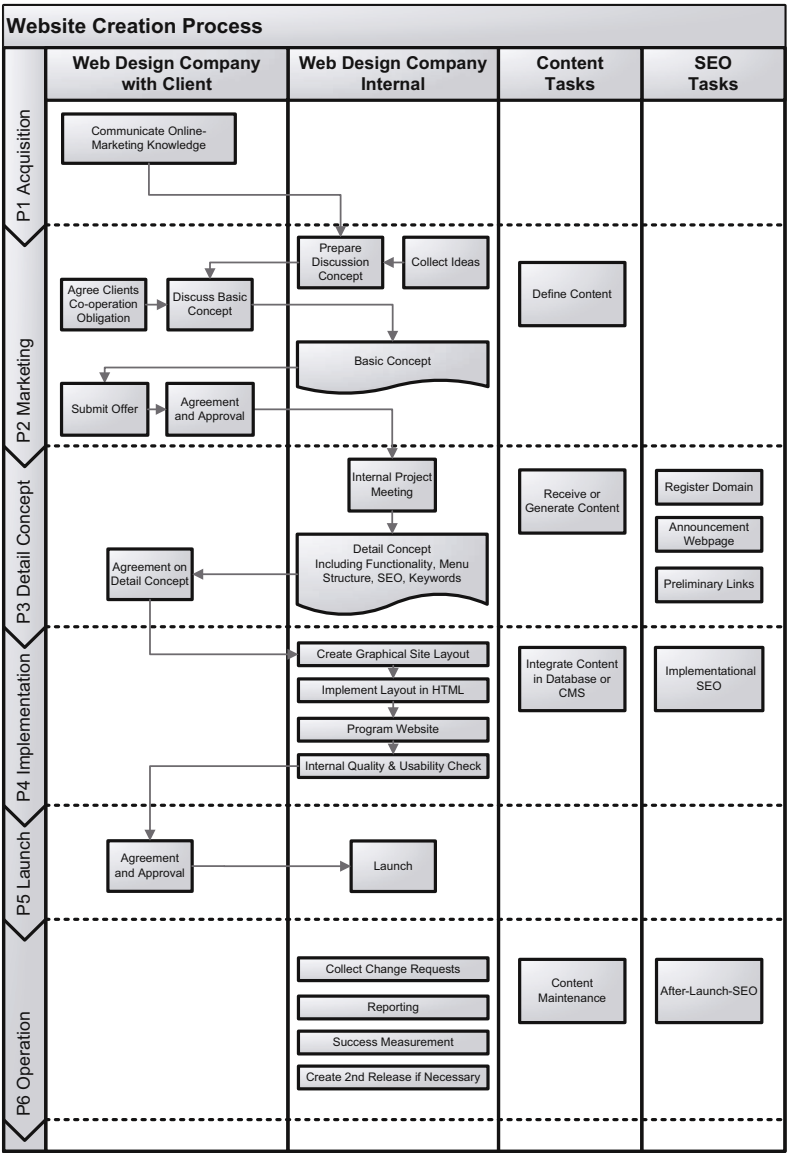


Figure 4.5: Website creation process

4.6 Practical Results

The effects of the possibilities and means of search engine optimization enumerated are tested for their practical suitability. Two existing website projects are subject for the following analysis. The applicability of the SEO task list introduced is measured against typical websites goals like visitor numbers and turnover. The listed means are simultaneously applied to the websites. Analyzing single means separately does not appear to be sensible because applying the different methods in common usually produces synergy and better effects. Moreover, as the tests were conducted on productive real-life websites, a step-by-step approach with different measures cannot be economically justified to the website owners.

In the following, the setup of a project for improving ranking positions is described. Websites with two different goals are subject of investigation. Finally, the results are analyzed over a time period with a length between two and three years.

4.6.1 Project and Test Websites

In co-operation with a company that offers Web services, two of their websites projects were identified and chosen for the test of the SEO measures. The company runs multiple Internet platforms. The websites create turnover from advertisement and shop sales. The amount of turnover depends on the number and quality of visitors that reach these sites. New customers often reach the websites coming through search engines.

No.	Project Phase	Tasks
1	Analysis	Workshop: Review of Current Situation Definition of Success Factors
2	Quick Win Implementation	Technical Quick Wins Content Quick Wins
3	In-depth Analysis	Client Analysis Competitor Analysis Technical Analysis Content Analysis Definition of Actions Planned
4	Implementation	Realization of Planned Actions

Table 4.7: Project phases

A project was established with the goal of increasing the number and quality of Web users reaching the websites as well as the amount of shop sales generated by existing and new customers. The project was funded by the Web service company and the European Social Fund (ESF). A project plan was created consisting of four phases (compare Table 4.7). During the project run time, the tasks of the different phases were concretized.

Analysis Phase
During the first phase, the *Analysis Phase*, a review of the current situation was performed. A workshop is conducted to gain an overview over existing clients, their projects and which services are offered to them. The goals of the different websites are analyzed. Success factors for the websites are defined. A first analysis of websites with respect to SEO is performed. For this reason, the access log files of the websites are analyzed. The following two websites are chosen for a close examination.

Websites
The first website regarded is a directory of suppliers in the leisure industry connected with geographic and cartographic information. Before the project started, the directory had about 30'000 page impressions (compare Table 4.1) per month. This corresponds to about 5'000 unique visitors monthly. The goal of the website is to attract as many visitors as possible to the website. The website owner receives money for directory entries by its customers and for advertisements placed on the directory pages. Thus, the website goal is to increase the number of page impressions and visitors (compare Figure 4.1).

The second website regarded is an online shop. It sells products for the outdoor leisure industry. It had realized a monthly turnover of several hundred thousand Euros with the online shop. The website owner is interested in increasing his turnover resulting from sales of the online shop. He differentiates between sales of new customers and existing customers. The business goal was to particularly increase the number and sales volume of new customers by SEO means. A subordinate goal was to increase the number of visitors on the website. Before the start of the project, about 500'000 page impressions were counted per month on the website. This corresponds to about 45'000 unique visitors monthly.

The goals of both websites can be approached by increasing their positions in the search-engine result-pages of appropriate keywords.

Quick Win Implementation Phase
Based on the results of the first analysis of the websites, those measures are identified that can be quickly implemented with low effort *(quick wins)*. These are

especially measures from the content area like missing keywords. In the technology area, e.g. meaningful URLs are introduced. The identified measures are implemented in this phase as far as possible.

In-depth Analysis Phase
The *In-depth Analysis Phase* comprises the analysis of different areas. The circumstances of the clients and their competitors are analyzed. The results of this analysis are already incorporated into the keyword definition. Besides a technical and content analysis, all areas of the SEO task list are regarded. Specific tasks for websites are defined. The process of website creation is captured and modeled. The SEO tasks are mapped to the different process steps as shown in Figure 4.5.

Implementation Phase
During the *Implementation Phase,* the defined tasks are put into practice. In order to quantify the success, some measure points are defined (see Report column of Figure 4.4) and reporting tools analyzing the user access to the regarded websites are implemented.

4.6.2 Reporting and Results

The effects of the SEO task list application are measured against different goals of the websites regarded.

For both, the directory and the online shop, the website goals of the SEO impact chain are relevant. The website goals are to increase the number of page impressions and the number of visitors on the website. Both numbers can be measured by tracking tools. For this analysis, the public available ETRACKER tool was chosen and implemented on the websites.

For the online shop, additional business goals are pursued. In order to increase the shop's profit, the total sales volume shall be boosted by attracting more visitors on the website. The sales volume can be increased by the number of sales or the turnover per sales transaction. For this reason, the number of sales is measured separately. The sales volume and the number of sales are captured by the internal accounting software of the shop.

Directory
The SEO tasks for the directory website were implemented from April to October 2006. The number of visitors and page impressions were already tracked before the implementation start (since May 2005). In the total measurement period from May 2005 until December 2007, about 2.5 millions of page impressions were

	May – Dec 2005	2006	2007	Total
Page Impressions	236’215	606’801	1’611’825	2’454’841
Visitors	39’384	182’050	247’254	468’688
Correlation between PI and Visitors	0.83	0.91	0.77	0.76

Table 4.8: Page impressions and visitors for the directory

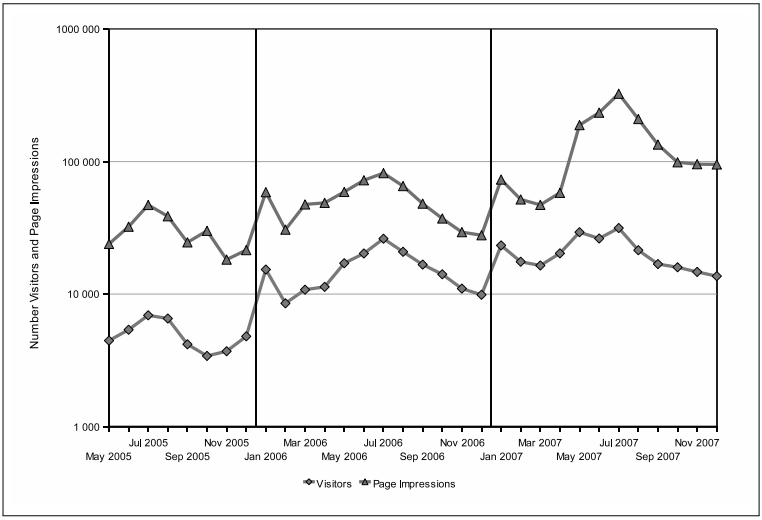


Figure 4.6: Directory – page impressions and visitors per month

counted (compare Table 4.8). They are generated by almost half a million visitors. An increase of both measures from 2006 to 2007 can already be observed.

The monthly figures in Figure 4.6 allow a more detailed analysis. The charts show the parallel development of visitors and page impressions on the directory from May 2005 until the year’s end of 2007. The values are linked by a high correlation of 0.76 over the whole measurement period. The correlations of the single years are slightly higher. All correlation values can be found in Table 4.8.

In both charts, a seasonal increase during the summer months can be observed. This can be explained by the nature of the directory’s topic. The directory holds information needed for outdoor leisure activities that are primarily practiced in sum-

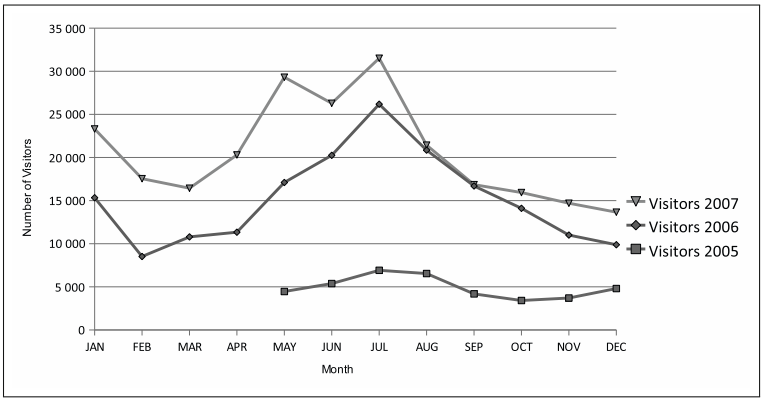


Figure 4.7: Directory – visitors per month over the courses of the years 2005–2007

	Jan–Oct 2007	Nov–Dec 2007	Jan–Oct 2008	Total
Page Impressions	5'375'569	797'077	6'805'202	12'977'848
Visitors	470'498	76'775	622'677	1'169'950

Table 4.9: Page impressions and visitors for the online shop

mer. Consequently, most people access the directory during the summer months. Another peak can be observed in each January, where a yearly exhibition on the same topic takes place.

As the seasonal variations of the visitor numbers can bias the measurement of SEO effects, the visitor numbers over the courses of the years are assembled in Figure 4.7. A comparison of the same months of different years allows a seasonally adjusted analysis of the visitor numbers. An increase of the visitor numbers in the summer of 2006 compared to the summer of 2005 is clearly visible. The chart of 2007 exhibits another increase in visitor numbers taking place compared to 2006.

Online Shop

For analyzing the effects on the online shop, page impressions and visitors on the website were captured from January 2007 to October 2008 using the ETRACKER tool. The turnover figures of the shop's internal accounting are available from January 2007 to November 2008.

Correlation between:	and:	Jan–Dec 2007	Jan–Oct 2008	Jan 2007 –Oct 2008
Page Impressions	Visitors	0.96	0.96	0.96
Visitors	Number of Sales	0.90	0.87	0.84
Visitors	Number of New Customers' Sales	0.88	0.90	0.88
Visitors	New Customers' Sales Volume	0.85	0.81	0.86
Number of New Customers' Sales	Number of Existing Customers' Sales	0.91	0.86	0.89
New Customers' Sales Volume	Existing Customers' Sales Volume	0.92	0.84	0.89

Table 4.10: Correlation coefficients for the online shop

The SEO tasks were implemented in the period from May 2007 until November 2007. In the observation period of the website, almost 13 million page impressions were counted (see Table 4.9). They were generated by nearly 1.2 million unique visitors. The number of monthly page impressions is strongly correlated with the visitor number (compare Table 4.10). The number of visitors on the shop's website depends on the season (like it is the case at the directory). For this reason, it is sensible to compare the same periods of different years. As the visitor numbers for 2008 are measured from January to October, they are compared with the numbers of January to October 2007. The visitor number has increased by about 30 %. An increase of the page impressions can be observed, too. A more detailed analysis on a monthly basis is performed with respect to the sales generated by the visitors, because the sales volume and the number of visitors are the final business goals pursued by the website owner.

In the total analysis period from January 2007 until November 2008, a turnover of more than three million Euros was achieved (see Table 4.11). Thirty thousand sales transactions have produced this turnover. This leads to an average sales volume of 109 € per transaction.

For a more detailed analysis of the effects, the customers are differentiated by three types. Those customers that have already bought via mail-order before the shop went online are called *existing customers*. They have produced more than 18'000 sales transaction with a total volume of more than two million Euros.

Sales of	Number of Sales	Sales Volume	Avg. Volume per Sale
Existing Customers	18'567	2'079'000 €	112 €
New Customers	8'255	855'000 €	104 €
Follow-up Customers	3'184	351'000 €	110 €
Total	30'006	3'285'000 €	109 €

Table 4.11: Sales of customer type for online shop for January 2007 – November 2008

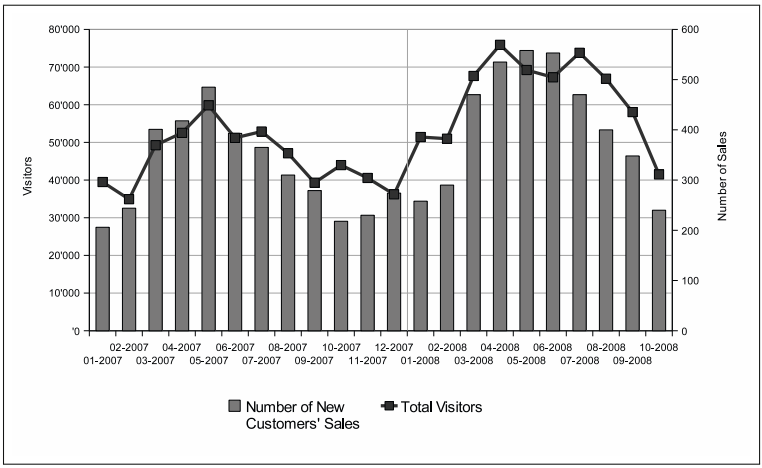


Figure 4.8: Shop – visitors and number of new customers' sales per month

Those customers that buy the first time in the online store and have not used the mail-order store before are called *new customers*. They make up roughly one third of the sales volume and the number of sales of existing customers. *Follow-up customers* constitute the third group. These are those new customers that buy again at the online shop. Their number of sales and sales volume is about one sixth of the values of the existing customers. All customer types have an average sales volume of a little more than one hundred Euros per transaction.

The most relevant customer group for the evaluation of the effects of the SEO measures is the group of new customers. Thus, the sales of new customers are analyzed in more detail. Figure 4.8 shows the number of sales performed by new customers for the single months of the observation period. The peaks in the summer months exhibit the seasonal character of the business. The number of sales

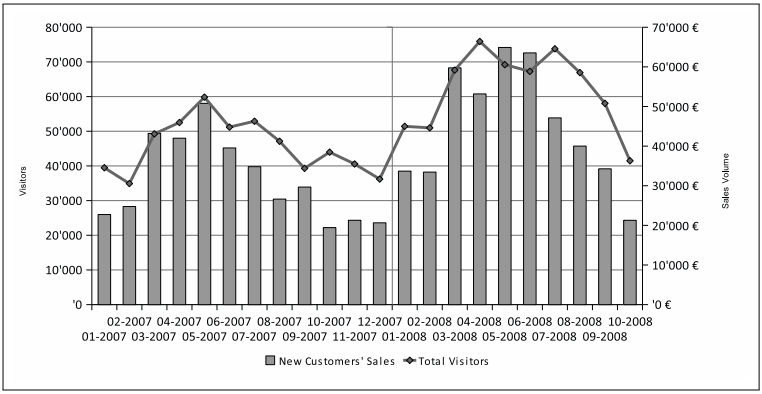


Figure 4.9: Shop – visitors and new customers' sales per month

also strongly correlates with the number of visitors on the website with correlation coefficients near 0.9 (compare Table 4.10). An increase of both, the numbers of visitors and of sales, is clearly visible in 2008 compared to 2007.

Similar observations can be made comparing the visitors with the new customers' sales per month (see Figure 4.9). These values are correlated with a correlation coefficient of more than 0.9. An increase of the sales volume of new customers in 2008 against 2007 is clearly visible. Together with the strong correlation of visitor numbers and sales volumes, this allows the conclusion that the visitor number positively influences the sales volume produced by new customers.

It is also possible that other external factors positively influence the sales volumes of new customers. This can e.g. be caused by a higher general disposition to buy, better weather conditions or different price structures. In order to exclude these factors from the analysis, it is assumed that external factors affect the sales of existing customers and new customers in the same way and by the same amount. This means that if existing customers buy more because of better weather then sales of new customers increase in the same relation. Calculating a quotient of new customers' sales and existing customers' sales eliminates the effect of weather or other external factors. This approach simultaneously implies an adjustment for seasonal variations.

Figure 4.10 shows the quota of new customers' sales volume and existing customers' sales volume for each month of the observation period. The sub-figures (a) and (b) oppose the development of the years 2007 and 2008. As there is not any

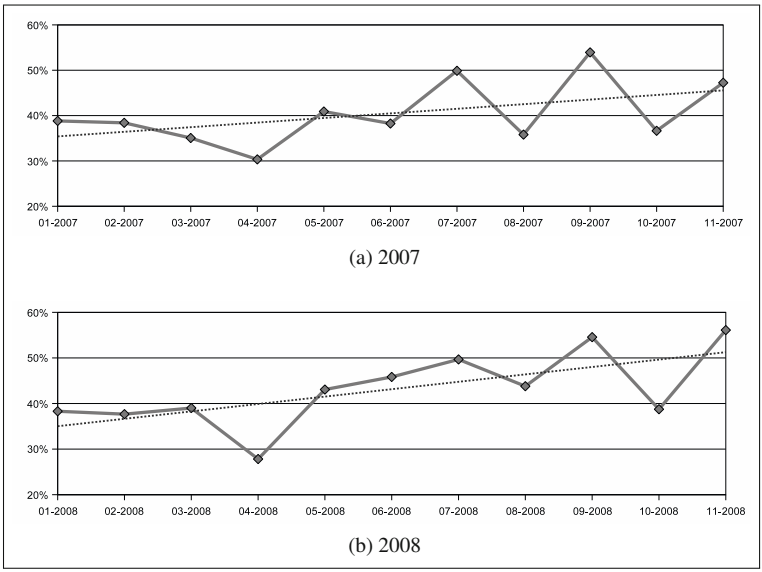


Figure 4.10: New sales quota with trend line

data captured for December 2008, also in this graph the December 2007 values are omitted.

Even though the sales volumes of existing and new customers are strongly correlated (compare Table 4.10), they diverge over time. Both values increase, but the trend line (calculated as linear regression) shows that the sales volumes of new customers stronger increase than those of existing customers. The trend line of the year 2008 is even steeper than the one of 2007. This implies (besides an absolute growth) also a relative growth of the new customers' sales volume. The fact that the sales volume generated by new customers grows stronger than the one of existing customers allows the conclusion that comparably more Web customers are generated in the observation period. As most new visitors come from search engines this indicates the positive effects of the SEO measures performed in the context of this work on the shop's sales volume.

5 Conclusion

For a user who is searching for information, it is of minor importance if this information *does not exist* in the Web or if this information only *cannot be found* in the Web. If the information is out of sight, the user cannot benefit from its existence. The important question is not anymore *where* to look for information, but *how* to find it. Here, the need for powerful search tools becomes visible.

This work contributes to both, the scientific and the practical evolution of Web search. New theoretical concepts are developed and successfully tested for practical usage.

The information demand of users is satisfied with different search tools. The VOX POPULI algorithm is developed in order to better meet the users' needs. It allocates crawler and storage resources according to the information demand. The importance of the VOX POPULI algorithm has already been shown by a wide resonance in the scientific community through several citations in journals and books. In the framework of future works, the opportunity can be taken to apply the algorithm in large commercial search engines.

The information supply is represented by the Web contents connected with hyperlinks. Extracting structures out of the Web graph contributes to the creation of appropriate search results. Own data structures are generated to store search results for the purpose of further processing. The data structures are implemented and tested for their applicability on actual search engine results. A suitable data structure was selected based on the criteria memory demand, performance and scalability. Its practical usability is proved upon an implementation of the HITS algorithm.

For matching the information need with the information supply, presentation methods are introduced that address the quality of search results. Different clustering algorithms are examined for their applicability on Web search. Combinations of several different clustering methods are implemented and tested on practical data. A comparison of their experimental results yields a method combination that is best suitable for improving result quality.

Finally, the influence that Web suppliers are able to exert on search results is exposed. Their objectives are analyzed and integrated in an impact chain, which shows how search engine optimization activities can support them in reaching their

economic goals. The interaction between website visitors and result positions is analyzed using mathematical notation. Three new keyword selection models are introduced to build a basis for search engine optimization efforts. Recommended efforts are described in detail resulting in a SEO task list that is implemented in a self-established website creation process. The effectiveness of the proposed measures is proved by positive developments of page impressions, visitors and sales volumes. Here, a deeper analysis of the impacts per single keywords can be subject of further work.

This work highlights several aspects of bringing order into the complexity of the World Wide Web by analyzing search engine results and the possibilities to influence these results. In different areas, the scientific perspective of business and technology impacts on Web information retrieval facilitates the achievement of better results in practice.

Bibliography

Aasheim, C. and Koehler, G. J. (2006). Scanning world wide web documents with the vector space model. *Decision Support Systems*, 42:690–699.

Abou-Assaleh, T., Das, T., Gao, W., Miao, Y., O'Brien, P., and Zhen, Z. (2007). A link-based ranking scheme for focused search. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1125–1126, New York, NY, USA. ACM.

Achlioptas, D., Fiat, A., Karlin, A. R., and McSherry, F. (2001). Web search via hub synthesis. In *IEEE Symposium on Foundations of Computer Science*, pages 500–509.

Adler, M. and Mitzenmacher, M. (2001). Towards compressing Web graphs. In *Data Compression Conference*, pages 203–212.

Agarwal, A. and Chakrabarti, S. (2007). Learning random walks to rank nodes in graphs. In *ICML'07*.

Ahlers, D. and Boll, S. (2008). Urban Web crawling. In *LOCWEB '08: Proceedings of the first international workshop on Location and the web*, pages 25–32, New York, NY, USA. ACM.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Alby, T. and Karzaunikat, S. (2007). *Suchmaschinenoptimierung*. Hanser Fachbuchverlag, 2nd edition.

Arasu, A., Cho, J., Garcia-Molina, H., Paepcke, A., and Raghavan, S. (2001a). Searching the Web. *ACM Transactions on Internet Technology*, 1(1):2–43.

Arasu, A., Novak, J., Tomkins, A., and Tomlin, J. (2001b). PageRank computation and the structure of the Web: Experiments and algorithms. Technical report, IBM Almaden Research Center.

Arnold, S. E. (2005). *The Google Legacy – How Google's Internet Search is Transforming Application Software*. Infonortics, Tetbury, England.

Backstrom, L., Kleinberg, J., and Kumar, R. (2009). Optimizing web traffic via the media scheduling problem. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 89–98, New York, NY, USA. ACM.

Baeza-Yates, R. (2004). Web usage mining in search engines. In Scime, A., editor, *Web Mining: Applications and Techniques*, chapter XIV, pages 307–321. Idea Group Publishing.

Baeza-Yates, R. A. and Ribeiro-Neto, B. A. (1999). *Modern Information Retrieval*. ACM Press / Addison-Wesley.

Baldi, P., Frasconi, P., and Smyth, P. (2003). *Modeling the Internet and the Web, Probabilistic Methods and Algorithms*. John Wiley & Sons, Ltd, Chichester, West Sussex, England.

Barabási, A.-L. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286:509–512.

Barabási, A.-L., Albert, R., and Jeong, H. (2000). Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A*, 281:69–77.

Becchetti, L., Castillo, C., Donato, D., Leonardi, S., and Baeza-Yates, R. (2006). Link-based characterization and detection of Web spam. In *AIRWeb*.

Bergman, M. K. (2001). The deep Web: Surfacing hidden value. Technical report, BrightPlanet.

Berkhin, P. (2006). A survey of clustering data mining techniques. In Kogan, J., editor, *Grouping Multidimensional Data*, pages 25–71. Springer, Berlin.

Berners-Lee, T. and Cailliau, R. (1990). WorldWideWeb: Proposal for a hypertext project. Technical report, CERN.

Berry, M. W. and Browne, M. (1999). *Understanding Search Engines: Mathematical Modeling and Text Retrieval*. SIAM.

Berry, M. W. and Castellanos, M., editors (2008). *Survey of Text Mining II. Clustering, Classification, and Retrieval*. Springer London, London, 1. edition.

Bharat, K., Broder, A., Henzinger, M., Kumar, P., and Venkatasubramanian, S. (1998). The connectivity server: fast access to linkage information on the Web. In *Proceedings of the 7th international World Wide Web Conference*, pages 469–477. Elsevier Science Publishers B. V.

Bharat, K., Chang, B.-W., Henzinger, M., and Ruhl, M. (2001). Who links to whom: Mining linkage between Web sites. In *Proceedings of the IEEE International Conference on Data*, pages 51–58.

Bharat, K. and Henzinger, M. R. (1998). Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval*, pages 104–111, Melbourne, AU.

Boldi, P. and Vigna, S. (2004). The webgraph framework i: compression techniques. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 595–602, New York, NY, USA. ACM.

Borodin, A., Roberts, G. O., Rosenthal, J. S., and Tsaparas, P. (2001). Finding authorities and hubs from link structures on the World Wide Web. In *Proceedings of the 10th international World Wide Web Conference*, pages 415–429.

Borthakur, D. (2007). The Hadoop distributed file system: Architecture and design. Technical report, The Apache Software Foundation, Delaware.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hyptertextual Web search engine. In *Proceedings of the 7th international World Wide Web Conference*, Brisbane, Australia.

Broder, A. (2002a). A taxonomy of Web search. *SIGIR Forum*, 36(2):3–10.

Broder, A. (2002b). Tutorial on search from the Web to the enterprise: Issues, solutions, evaluation I–II. Tutorial held at the ACM SIGIR Conference, Tampere (2002).

Broder, A. (2006a). From query based information retrieval to context driven information supply. Technical report, Yahoo! Research.

Broder, A. (2006b). The future of Web search: From information retrieval to information supply. *Lecture Notes in Computer Science*, 4032:362.

Broder, A., Kumar, R., Maghoul, F., Raghavan, P., and Stata, R. (2000). Graph structure in the Web. In *Proceedings of the 9th International World Wide Web Conference*, pages 247–256. ACM.

Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. (1997). Syntactic clustering of the Web. *Comput. Netw. ISDN Syst.*, 29(8-13):1157–1166.

Bronstein, I. N. and Semedjajew, K. A. (1981). *Handbook of Mathematics*. Nauka.

Castillo, C., Chellapilla, K., and Davison, B. D. (2008). Adversarial information retrieval on the Web (AIRWeb 2007). *SIGIR Forum*, 42(1):68–72.

Castillo, C., Marin, M., Rodríguez, A., and Baeza-Yates, R. (2004). Scheduling algorithms for Web crawling. In *Latin American Web Conference (WebMedia/LA-WEB)*, pages 10–17, Riberao Preto, Brazil. IEEE CS Press.

Chakrabarti, S. (2003). *Mining the Web*. Morgan Kaufmann Publishers, San Francisco.

Chakrabarti, S., Dom, B. E., and Indyk, P. (1998). Enhanced hypertext categorization using hyperlinks. In Haas, L. M. and Tiwary, A., editors, *Proceedings of SIGMOD-98, ACM International Conference on Management of Data*, pages 307–318, Seattle, US. ACM Press, New York, US.

Chakrabarti, S., Dom, B. E., Kumar, S. R., Rajagopalan, P. R. S., Tomkins, A., Gibson, D., and Kleinberg, J. (1999a). Mining the Web's link structure. *Computer*, 32(8):60–67.

Chakrabarti, S., van den Berg, M., and Dom, B. (1999b). Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1623–1640.

Cho, J. and Garcia-Molina, H. (2000). The evolution of the Web and implications for an incremental crawler. In *Proceedings of the Twenty-sixth International Conference on Very Large Databases*, pages 200–209.

Cho, J. and Garcia-Molina, H. (2003). Estimating frequency of change. *ACM Transactions on Internet Technology*, 3(3):256–290.

Cho, J., Garcia-Molina, H., and Page, L. (1998). Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172.

Chung, D. and Klündler, A. (2007). *Suchmaschinen-Optimierung*. mitp-Verlag.

Clay, B. (2009). Search engine relationship chart. www.bruceclay.com/searchenginerelationshipchart.htm (07/2002, 07/2004, 04/2006, 10/2009).

Cohn, D. and Chang, H. (2000). Learning to probabilistically identify authoritative documents. In *Proceedings of the seventeenth International Conference on Machine Learning*, pages 167–174. Morgan Kaufmann, San Francisco, CA.

Cohn, D. and Hofmann, T. (2001). The missing link – a probabilistic model of document content and hypertext connectivity. In *Neural Information Processing Systems 13*, pages 430–436.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press, 2nd edition.

Cutting, D. (2005). Scalable computing with MapReduce. Session held on the OSCON 2005, Portland, OR.

Cutting, D. R., Pedersen, J. O., Karger, D., and Tukey, J. W. (1992). Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 318–329.

Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. *OSDI '04*, pages 137–150.

Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

Dean, J. and Henzinger, M. R. (1999). Finding related pages in the world wide web. In *Proceeding of the eighth international conference on World Wide Web*, pages 1467–1479. Elsevier North-Holland, Inc.

Deerwester, S. C., Dumais, S. T., Furnas, G. W., Landauer, T., and Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal Royal Statist. Soc., Series B*, 39:1–38.

DENIC (2008). Die DENIC macht das Dutzend voll. Press Release 14.04.2008, DENIC, Frankfurt/Main.

Deo, N. and Gupta, P. (2001a). Graph-theoretic Web algorithms: An overview. *Lecture Notes in Computer Science*, 2060:91–102.

Deo, N. and Gupta, P. (2001b). World Wide Web: A graph-theoretic perspective. Technical report, CSTR-01-001, School of Computer Science, University of Central Florida, Orlando, FL.

Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., and Gori, M. (2000). Focused crawling using context graphs. In *26th International Conference on Very Large Databases, VLDB 2000*, pages 527–534, Cairo, Egypt.

Dorndorf, U. and Pesch, E. (1994). Fast clustering algorithms. *ORSA Journal on Computing*, 6:141–153.

Erdös, P. and Rényi, A. (1960). On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academie of Sciences*, 5:17–61.

Erlhofer, S. (2007). *Suchmaschinen-Optimierung*. Galileo Computing, 3rd edition.

Fallows, D. (2005). Search engine users: Internet searchers are confident, satisfied and trusting – but they are also unaware and naïve. Technical report, Pew Internet & American Life Project, Washington DC.

Fallows, D. (2008). Search engine use: Almost half of all internet users now use search engines on a typical day. Technical report, Pew Internet & American Life Project, Washington DC.

Flake, G. W., Lawrence, S., Giles, C. L., and Coetzee, F. (2002). Self-organization of the Web and identification of communities. *IEEE Computer*, 35(3):66–71.

Ford, L. and Fulkerson, D. (1956). Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404.

Frakes, W. B. (1992). Stemming algorithms. In Frakes, W. B. and Baeza-Yates, R., editors, *Information Retrieval: Data Structures & Algorithms*, pages 131–160. Prentice-Hall, Englewood Cliffs, NJ.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The Google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA. ACM Press.

Glöggler, M. (2003). *Suchmaschinen im Internet*. Springer.

Google (2009). Webmaster guidelines. http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=35769, downloaded 24th February 2009.

Grefenstette, G. (1995). Comparing two language identification schemes. In *Proceedings of the 3rd International Conference on the Statistical Analysisof Textual Data*, pages 263–268, Rome.

Guha, S., Rastogi, R., and Shim, K. (2000). ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366.

Gyöngyi, Z., Berkhin, P., Garcia-Molina, H., and Pedersen, J. (2006). Link spam detection based on mass estimation.

Henzinger, M. R., Motwani, R., and Silverstein, C. (2002). Challenges in Web search engines. *SIGIR Forum*, 36(2).

Hofmann, T. (1999a). Probabilistic latent semantic analysis. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 289–296, Stockholm.

Hofmann, T. (1999b). Probabilistic latent semantic indexing. In *Research and Development in Information Retrieval*, pages 50–57.

Hofmann, T. (2000). Learning probabilistic models of the Web. In *Research and Development in Information Retrieval*, pages 369–371.

Hofmann, T. (2001). From bits to information – Maschinelle Lernverfahren in Information Retrieval und Web Mining. Invited Colloquium Presentation at the University of Bonn, 17th July 2001.

Huberman, B. A. and Adamic, L. A. (1999). Growth dynamics of the World Wide Web. *Nature*, 401:131.

Jansen, B. J., Booth, D. L., and Spink, A. (2007). Determining the user intent of Web search engine queries. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1149–1150, New York, NY, USA. ACM Press.

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

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

Kleinberg, J. M., Kumar, R., Raghavan, P., and Tomkins, S. R. A. S. (1999). The Web as a graph: Measurements, models and methods. *Lecture Notes in Computer Science*, 1627:1–17.

Koster, M. (1995). Robots in the Web: threat or treat? *ConneXions*, 9(4).

Kotler, P. and Keller, K. L. (2008). *Marketing-Management*. Prentice Hall, 13th edition.

Kumar, R., Raghavan, P., Rajagopalan, S., Sivakumar, D., Tompkins, A., and Upfal, E. (2000). The Web as a graph. In *Proceedings of the nineteenth ACM symposium on Principles of database systems*, pages 1–10, New York, NY, USA. ACM.

Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. (1999). Trawling the Web for emerging cyber-communities. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1481–1493.

Lawrence, S. and Giles, C. L. (1998). Searching the World Wide Web. *Science*, 280(5360):98–100.

Lewandowski, D. (2008). *Handbuch Internet-Suchmaschinen*. AKA, Heidelberg.

Lewandowski, D., Wahlig, H., and Meyer-Bautor, G. (2006). The freshness of Web search engine databases. *Journal of Information Sciences*, 32(2):131–148.

Lieberam-Schmidt, S. and Pesch, E. (2008). Clustering of Web search results. *International Journal of Information Technology and Intelligent Computing: IT&IC*, 3(3).

Lovins, J. B. (1968). Development of a stemming algorithm. *Mechanical translation and computational linguistics*, 11:22–31.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

McCarthy, J. (1975). *Basic Marketing - a managerial approach*. Irwin.

Michael, M., Moreira, J. E., Shiloach, D., and Wisniewski, R. W. (2007). Scale-up x scale-out: A case study using Nutch/Lucene. In *Parallel and Distributed Processing Symposium, 2007*, pages 1–8. IEEE International.

Milgram, S. (1967). The small world problem. *Psychology Today*, 61:60–67.

Modha, D. S. and Spangler, W. S. (2000). Clustering hypertext with applications to Web searching. In *ACM Conference on Hypertext*, pages 143–152.

Moore, A. (1999). Very fast EM-based mixture model clustering using multiresolution kd-trees. In *Advances in Neural Information Processing Systems 11*, pages 543–549. MIT Press.

Moreira, J. E., Michael, M. M., Silva, D. D., Shiloach, D., Dube, P., and Zhang, L. (2007). Scalability of the Nutch search engine. In *ICS '07: Proceedings of the 21st annual international conference on Supercomputing*, pages 3–12, New York, NY, USA. ACM.

Motwani, R. and Raghavan, P. (1995). *Randomized Algorithms*. Cambridge Univ. Press.

Najork, M. A., Zaragoza, H., and Taylor, M. J. (2007). HITS on the Web: how does it compare? In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 471–478, New York, NY, USA. ACM.

Newman, M., Moore, C., and Watts, D. (2000). Mean-field solution of the small-world network model. *Physical Review Letters*, 84:3201–3204.

Ntoulas, A., Najork, M., Manasse, M., and Fetterly, D. (2006). Detecting spam Web pages through content analysis. In *Proceedings of the 15th International World Wide Web Conference*, pages 83–92.

Osinski, S. (2003). An algorithm for clustering of Web search results. Master's thesis, Poznan University of Technology, Poland.

Osinski, S., Stefanowski, J., and Weiss, D. (2004). Lingo: Search results clustering algorithm based on singular value decomposition. In *Intelligent Information Systems*, pages 359–368.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford Digital Library Technologies Project.

Pennock, D. M., Flake, G. W., Lawrence, S., Glover, E. J., and Giles, C. L. (2002). Winners don't take all: Characterizing the competition for links on the web. In *Proceedings of the National Academy of Sciences*, pages 5207–5211.

PEW (2007). December 2006 tracking survey. Technical report, Princeton Survey Research Associates International for the Pew Internet & American Life Project.

PEW (2009). Daily internet activities, 2000–2009. www.pewinternet.org/Trend-Data/Daily-Internet-Activities-20002009.aspx.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–137.

Rafiei, D. and Mendelzon, A. O. (2000). What is this page known for? computing Web page reputations. In *Proceedings of the 9th international World Wide Web conference on Computer networks*, pages 823–835, Amsterdam, The Netherlands. North-Holland Publishing Co.

Raghavan, S. and Garcia-Molina, H. (2001). Crawling the hidden Web. In *Proceedings of the Twenty-seventh International Conference on Very Large Databases*, pages 129–138.

Raghavan, S. and Garcia-Molina, H. (2003). Representing Web graphs. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 405–416.

Rainie, L. and Shermak, J. (2005). Search engine use november 2005. Technical report, PEW Internet & American Life Project, Washington, DC.

Randall, K. H., Stata, R., Wiener, J. L., and Wickremesinghe, R. G. (2002). The link database: Fast access to graphs of the web. In *DCC '02: Proceedings of the Data Compression Conference (DCC '02)*, page 122, Washington, DC, USA. IEEE Computer Society.

Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., and Kozyrakis, C. (2007). Evaluating mapreduce for multi-core and multiprocessor systems. In *HPCA '07: Proceedings of the 13th International Symposium on High-Performance Computer Architecture*, pages 13–24. IEEE Computer Society.

Risvik, K. M. and Michelsen, R. (2002a). An overview of search engine technology – part I: Foundations. Technical report, Fast Search & Transfer ASA, Trondheim.

Risvik, K. M. and Michelsen, R. (2002b). Search engines and Web dynamics. *Computer Networks*, 39(3):289–302.

Rose, D. E. and Levinson, D. (2004). Understanding user goals in Web search. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 13–19, New York, NY, USA. ACM Press.

Rose, K., Gurewitz, E., and Fox, G. (1990). A deterministic annealing approach to clustering. *Pattern Recognition Letters*, 11(9):589–594.

Saito, H., Toyoda, M., Kitsuregawa, M., and Aihara, K. (2007). A large-scale study of link spam detection by graph algorithms. In *AIRWeb 07*, pages 45–48.

Salton, G. (1989). *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, Reading, MA.

Saul, L. and Pereira, F. (1997). Aggregate and mixed-order Markov models for statistical language processing. In Cardie, C. and Weischedel, R., editors, *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 81–89, Somerset, New Jersey. Association for Computational Linguistics.

Schaale, A., Wulf-Mathies, C., and Lieberam-Schmidt, S. (2003). A new approach to relevancy in internet searching – the "Vox Populi Algorithm". *CoRR*, cs.DS/0308039.

Schwarz, T. (2008). *Leitfaden Online Marketing*. Marketing Börse, 2nd edition.

Sherman, C. (2005). A new F-word for Google search results. *Search Engine Watch*.

Skiera, B. and Spann, M. (2000). Werbeerfolgskontrolle im Internet. *Controlling*, 12:417–423.

Smyth, P. (1996). Clustering using monte-carlo cross validation. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 126–133, Portland, OR. AAAI Press.

Stata, R., Bharat, K., and Maghoul, F. (2000). The term vector database: fast access to indexing terms for Web pages. In *Proceedings of the 9th international World Wide Web conference on Computernetworks*, pages 247–255. North-Holland Publishing Co.

Steinbach, M., Karypis, G., and Kumar, V. (2000). A comparison of document clustering techniques. In *6th ACM SIGKDD, World Text Mining Conference, Boston, MA.*, pages 109–110.

Suel, T. and Yuan, J. (2001). Compressing the graph structure of the Web. In *Data Compression Conference*, pages 213–222.

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

White, R. W. and Drucker, S. M. (2007). Investigating behavioral variability in Web search. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 21–30, New York, NY, USA. ACM.

White, T. (2006). Introduction to Nutch, part 1: Crawling. Technical Report 01/10/2006, Sun Microsystems, Inc.

Zamir, O. and Etzioni, O. (1998). Web document clustering: A feasibility demonstration. In *Research and Development in Information Retrieval*, pages 46–54.

Zamir, O. and Etzioni, O. (1999). Grouper: a dynamic clustering interface to Web search results. In *WWW '99: Proceeding of the eighth international conference on World WideWeb*, pages 1361–1374, New York, NY, USA. Elsevier North-Holland, Inc.

Zhang, D. and Dong, Y. (2000). An efficient algorithm to rank web resources. *Computer Networks*, 33(1-6):449 – 455.

Zhang, D. and Dong, Y. (2004). Semantic, hierarchical, online clustering of Web search results. In *APWeb*, pages 69–78.

Index