University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Web Mining and Its Applications to Researchers Support

State of the Art & Concepts of Doctoral Thesis

Dalibor Fiala

Technical Report No. DCSE/TR-2005-06
April, 2005

Distribution: public

Technical Report No. DCSE/TR-2005-06
April 2005

Web Mining and Its Applications to Researchers Support

Dalibor Fiala

Abstract

Web mining is a newly emerging research area concerned with analyzing the World Wide Web. It is concerned mainly with its content, structure and usage. As the Web is the largest storehouse of knowledge of our time it has become essential to learn how to exploit its potential. This report aims at presenting Web mining in the context of other scientific domains and it introduces its possible applications to the support of researchers. The areas discussed are numerous: data mining, topic detection, computational linguistics, ontologies, Web search engines, etc. Special attention is paid to Web page ranking methods. We also present our experiments with two algorithms for discovering repeated word sequences in texts. Throughout this report we underline deficiencies in the state-of-the-art knowledge which could serve as guidelines in our future research. Finally, we enumerate existing systems for research staff support and we sketch out a vision of the doctoral thesis.

This work has been supported by the Ministry of Education of the Czech Republic – project FR VS 1347/2005/G1.

Copies of this report are available at `http://www.kiv.zcu.cz/publications/` or by surface mail on request sent to the following address:

University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

University of West Bohemia in Pilsen
Faculty of Applied Sciences
Department of Computer Science and Engineering

University Louis Pasteur - Strasbourg I
National Institute of Applied Sciences
Laboratory of Informatics and Artificial Intelligence

Web Mining and Its Applications to Researchers Support

State of the Art
&
Concepts of Doctoral Thesis

Dalibor Fiala

April, 2005

Acknowledgements

I would like to thank my supervisors Karel Jezek and François Rousselot for enabling me pursuing a PhD degree under joint supervision both in the Czech Republic and in France. Both of them have helped me a lot overcome administrative obstacles. The latter has done especially much effort in searching for ways to finance my first stay in Strasbourg. Also, I would like to thank my colleague from the Text Mining Group Roman Tesar for his fruitful cooperation on the subject of repeated sequences.

Strasbourg, 19 April, 2005

Table of Contents

List of Figures

List of Tables

I Introduction

This report is an attempt at capturing the state of the art in the domain of Web mining and related techniques with a focus on extracting knowledge appropriate for researchers. We will present a brief overview of knowledge discovery in databases (*KDD*), address the problem of topic detection and particularly its linguistic aspects, pay some attention to link analysis, search engines and Web page ranking schemes, describe a couple of existing systems in our field of interest, and underline the need for a free tool for researchers. Finally, we will point out where there is some space for some research and we will sketch out fundamental concepts of the future Ph.D. thesis.

I.1 Knowledge Discovery in Databases

Knowledge discovery in databases (*KDD*) and data mining are sometimes confused with each other. Some authors, indeed, consider *KDD* and data mining as synonyms and use both terms arbitrarily. Others, however, regard *KDD* as a more general process, part of which is data mining. We prefer the latter concept because it complies better with the mining hierarchy in which we involve text mining and Web mining as well. Thus, starting from the most general mining the hierarchy looks like this: *KDD -> data mining -> text mining*. It is not evident where to put *Web mining*. Obviously Web is more than text. But is it a child of data mining or *KDD*? Whichever mining we mean, it is always a process of extracting hidden information, useful knowledge or interesting relations from some data. Obviously, the nature of this data determines the hierarchy level at which we "mine".

Let us now recall a few definitions of data mining:

"The nontrivial extraction of implicit, previously unknown, and potentially useful information from given data." [Piatetsky-Shapiro91]

"The science of extracting useful information from large data sets or databases." [Hand01]

"The extraction of interesting (non-trivial, implicit, previously unknown and potentially useful) information or patterns from data in large databases." [Han00]

And also some explanation of what it is all good for:

We are drowning in information, but starving for knowledge! (John Naisbett)

Let us remark that the last citation does not speak about data. But saying "We are drowning in data, but starving for information!" would make exactly the same sense. In either case, we have plenty of something and we would like to get little pieces of something else out of that. There is a similar relationship between data and information like between information and knowledge. The transition from data to information is made by processing data and, similarly, by processing information we move from information to knowledge. Thus, we may conclude that information is processed data and knowledge is processed information.

As we see, there is really some resemblance to a real-world gold mining, for instance. Han [Han00] remarks in this context, that we should say *information mining* rather than data mining in order to conform to the "real-world" terminology. A gold miner mines gold (a precious metal) from some raw mould, which is not useful as such. And so we do – we mine

information (gold) from data (mould). To retain *data* and avoid misunderstanding some authors prefer saying *mining from data*. However, the term *data mining* is broadly accepted and we will use it as well.

I.2 Data Mining

Data mining, as part of *KDD*, which involves some pre- and post-processing in addition (see Figure I.1), is closely related to other areas of computer science such as databases, artificial intelligence, information retrieval, machine learning, computational linguistics (as we will try to show), knowledge acquisition and knowledge engineering, decision support, ontologies… One of its finest applications is the *market basket analysis*, more thoroughly described in [Han00], for instance. We will only give a short introduction to this problem.

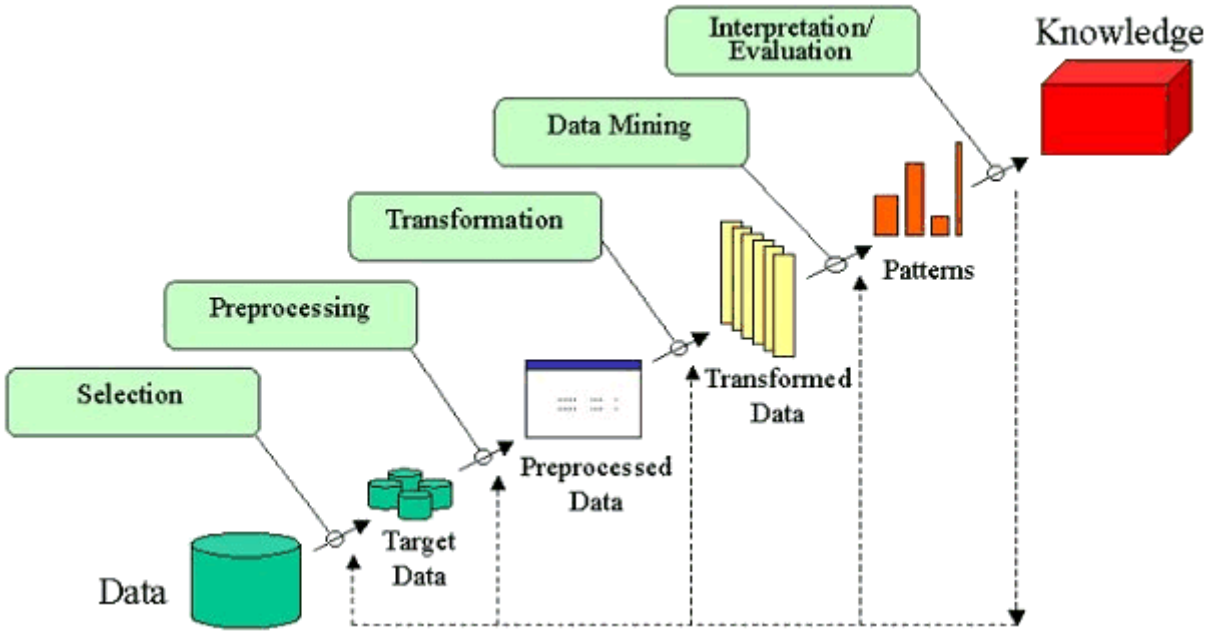

Figure I.1: Data mining as part of KDD process (from [1])

I.2.1 Market Basket Analysis

Imagine a customer leaving a shopping mall with a shopping basket (nowadays, rather with a car, though) full of goods. We will consider this basket as a *transaction* in a *transaction database*. Each *item* of goods (sugar, milk, bread…) is a transaction item. And there is another customer in whose basket there are cigarettes, beer and bread, so its transaction is {cigarettes, beer, bread}. We may expect that the total number of customers of a shopping mall could be in thousands or even in tens of thousands a day. A transaction database with several tens of thousands of records would certainly be a matter of interest of a marketing analyst of this shopping mall. For instance, the analyst could be interested in the probability that a person, who has bought beer, buys cigarettes as well (in the same transaction). Or in the ratio of transactions containing both beer and cigarettes to all transactions.

Here we come slowly to *association rules*. An *association rule* describes some relation (or association) between items in the *transaction database*. The rule "A customer, who has bought beer, buys cigarettes as well" could then be logged simply as {beer} -> {cigarettes}. There are two major indicators for each association rule – *confidence* and *support*. The first is its probability and the latter its relative count, expressed like

$$s(A \rightarrow B) = \frac{count(A \cup B)}{n} \quad (1)$$

$$c(A \rightarrow B) = \frac{s(A \rightarrow B)}{s(A)} \quad (2)$$

where s(A $\rightarrow$ B) is the *support* of the rule A $\rightarrow$ B (for instance, {beer} $\rightarrow$ {cigarettes}), c(A -> B) is the rule's *confidence*, *count*(A $\cup$ B) is the number of transactions containing A and B, and s(A) is the ratio of the number of transactions containing A to the number of all transactions n in the database.

Before processing the transaction database the analyst usually determines some threshold values for both *confidence* and *support* so as to eliminate uninteresting rules. The other rules, whose confidence as well as support lie above these thresholds, and which are therefore supposed to be interesting, will be called *strong association rules*.

Mining strong association rules from transaction databases is a fundamental feature of modern data mining. The basic mining algorithm is the *a priori algorithm.* This algorithm takes advantage of the a priori known fact that a *frequent n-itemset* cannot contain a non-frequent n-1-itemset. An *itemset* is a set of transaction items such as {bread, milk}. We may think of it as a helper (fictitious) transaction. The itemset {bread, milk} is a 2-itemset because it has two items. An itemset is frequent when its relative count is greater than or equal to the threshold support. So the algorithm starts generating frequent 1-itemsets then it continues producing frequent 2-itemsets (by pruning candidate itemsets using the prior knowledge above) and so on. The algorithm stops when there are no new frequent itemsets to obtain. Finally, it derives corresponding association rules from those frequent itemsets that satisfy the *confidence* constraint ([Rajman98]).

The association rules of type {beer} $\rightarrow$ {cigarettes}, with which we work in our example, are simple one-dimensional associations. All items there are goods. [Han00] gives examples of multi-dimensional and other more complicated association rules, which take account of a customer's age, address, abstraction levels of a product (e.g. a printer, a USB printer, an HP printer), etc. He also enumerates many possible modifications of the basic *a priori algorithm*, which consist mainly in parallelization, usage of special data structures (e.g. FP-trees), frequent itemsets pruning and other enhancements.

The analyst can take many conclusions analyzing the *strong association rules* mined from the *transaction database*. Not all of them may be interesting in the end. But some of them might significantly improve the shopping mall's performance when taken advantage of. For instance, if a *strong association rule* says that beer and cigarettes are always bought together, these goods could be placed in opposite corners of the store so that customers moving from one corner to the other could also buy other products. Or, on the contrary, they could be put next to each other to encourage customers to purchase them. For a decision maker, it is merely a matter of creativity, how he treats the information that has suddenly emerged from the data.

I.3 Text Mining

Text mining is a new specialized domain of *data mining* arising in the 1990s. The difference is that whereas *data mining* is concerned with (mostly structured) general type data, *text mining* uses similar tools to mine information from (mostly unstructured) text sources. These sources may be text corpora of random speech samples recorded in the street, documents in a

computer folder, e-mails, listings of computer directories, whatever documents downloaded from the Web, etc. It is interesting that once a document becomes structured (by transforming it from a plain text file into an XML document, for example), it moves to the realm of *data mining*. In fact, XML documents are logically or semantically structured, i.e. we can localize fragments in them knowing exactly what the contents of those fragments mean. On the other hand, finding a postal ZIP code in an e-mail would turn out more difficult. Besides, there exist semi-structured documents, such as the International Genealogical Index GEDCOM [39]. Briefly, the less structured documents we deal with the more appropriate is the usage of *text mining* techniques.

The father of *text mining* is Ronen Feldman ([Feldman95], [Feldman96], [Feldman97a] [Feldman97b], who introduced association rules mining at the level of keyword co-occurrences, among others. This *mining at the term level* ([Feldman98a], [Feldman98b]) has proven to bring even more useful and interesting relations and patterns. Terms in this sense have the same meaning as *repeated segments* about which we will talk later on. Ronen Feldman stands behind some commercial data and text mining applications in which he has employed the methods presented in his papers, such as ClearResearch discussed in Section V. *Text mining* is a new, challenging area of research whose possible applications ranging from topic detection, concepts discovery and collaboration networks analysis to detection of criminal activities are still not fully explored.

I.4 Web Mining

Web is a huge storehouse of potential knowledge. Shortly after its appearance evident problems arose of extracting from it the information we needed. The trouble is that it is vast (up to 5×10^{10} documents so far), volatile (it grows constantly, but at the same time, a significant number of documents disappear every day), heterogeneous (it preserves all kinds of documents – text, image, video, sound…), unstructured (apart from XML files and the like, Web documents are mostly highly unstructured) and the system of hyperlinks connecting one Web document with another one is a little labyrinth. It was soon discovered that not only the content of Web was important but also its topology. Although close to other "mining brothers", the notion of *Web graph* of documents connected via links makes *Web mining* deal with discrete mathematics (combinatorics and graph theory). [Chakrabarti02] presented perhaps the most comprehensive state of the art of Web mining so far.

We can observe that Web mining completes three main tasks: crawling, topology analysis and content analysis. ([Kosala00] adds usage analysis but neglects crawling.) Crawling is the art of traversing the Web graph efficiently and effectively. Chakrabarti proposes parallel methods that take account of server load balancing and enable avoiding *Web traps*, i.e. infinite cycles of documents linking to each other. Optimizing these crawling programs (*crawlers*, *spiders*, *Web robots*, *bots* or *agents*) is also an important factor in the design of Web search engines. These engines use *crawlers* to download Web documents for indexing. We will devote a special chapter to search engines (Section IV). Topology analysis consists in exploring relationships among Web sites so as to find the most authoritative sources on the Internet, for example. This may further enhance search engines' capabilities. Content analysis employs similar methods as data mining and machine learning, which we mention briefly in the next paragraph.

I.5 Machine Learning

The background idea behind all computerized activities is to automatize certain processes that would otherwise be done by humans with much more effort and in much more time. In order

to make machines "think" and do what we demand they must be trained first. In the case of *supervised learning* it is humans who teach them. For instance, the problem of *classification* (or *categorization*) of input objects can be solved by supplying the machine with a training set of pre-classified objects (where the objects were classified by humans), letting it learn classification rules and consequently getting it classify real samples. Let us remind that a *class* is a set of objects having the same or similar *attributes* (*features*). In *unsupervised learning*, such as *clustering*, it is the machines themselves that group samples into *clusters* in order to minimize intracluster and maximize intercluster distances. Letting alone the process of their creation, clusters and classes differ from each other only in that a cluster does not need to have a name (label) whereas each class is human labeled before. [Han00] and [Chakrabarti02] speak also about methods of *semi-supervised learning* that are a combination of both.

Further below we will stress the need for adaptive, semi-supervised machine learning techniques by which we teach the machines what is desirable and what is not, let them do their job, make corrections in case they make mistakes and, iteratively, we get them improve their performance. These methods are by far not so well documented as those in the first paragraph of this section.

I.6 Personalized Web Surveillance

The struggle for exploiting the full potential of the Web will probably go on in the next decades. What is more likely to become the reality of present days is the concept of personalized Web. *Personalized Web* is a Web tailored for a particular user. In the optimal case, a user works with a small part of Web only. With a part that satisfies his needs for information. This window into the whole World Wide Web is based on the user's profile taking account of his Web usage history (sites visited, pages bookmarked, time passed while looking at certain pages, etc.) and some additional information such as preferences set by user. Personalization allows for abstracting from the entire Web, reducing its size to much more manageable dimensions, thus facilitating the search for information and knowledge extraction.

We must underline in this context that *personalized Web* should not be confused with *semantic Web*. Tim Berners-Lee considers *semantic Web* as a place where humans as well as machines can exchange information. Data are semantically tagged to indicate what they represent in such a way that machines can process these tags. However, the tagging does not say how the machines should process it.

Programs that systematically search the Web and process the information found in a manner more or less independent of the end users are called *autonomous agents*, *intelligent agents* or simply *Web agents*. We may give an example of an *intelligent agent* usage. Suppose a businessman would like to take part in a business meeting held in a month on another continent. He wishes to get to the meeting and back in time and he would also like to minimize travel costs. He sends his request and constraints to an autonomous agent and he waits for its response. After some time, the *Web agent* replies with a list of, hopefully, cheap flight connections and it will have made a booking perhaps. The businessman has no clue how the agent acquired the results presented, whether it just searched the Web and filled in some forms, or whether it interacted with humans or even negotiated with other *intelligent agents*. The area of *autonomous Web agents* has an enormous potential that is yet to be fully exploited. It is a new research field with many open issues.

In this context, we might easily imagine a scientist wishing to have state-of-the-art knowledge in his or her field of interest. For this purpose, we could charge an intelligent agent with the

task of continuously watching over Web sites of other scientists and supplying our researcher with information on new research topics, publications, conferences, patents, newly emerged Web sites in a given domain, etc. This kind of *Web surveillance* is sometimes called *scientific watch* or *technological watch*. There are systems of scientific watch like Tétralogie but they do not work completely autonomously. See Section V.2 for more information on Tétralogie.

II Topic Detection

Topic detection is very close to classification (categorization) in that it labels documents so that a reader could characterize them with a few words without having to read them. The difference is that unlike classification *topic detection* deals with a more or less continuous feed of information where the topic (or topics) evolves in time. A typical example would be a news television Web site updating its content every time when needed, typically a couple of times per hour. There even exists the RSS meta-language [2] which is particularly useful for steady news feeds. RSS (RDF Site Summary) is based on XML and it specifies the way an emitter announces updates of its content so that a receiver could react accordingly. In this manner, a favourite Web browser is always capable of offering its users the latest news headlines. Another difference from classification is that results of this process are much less specified. We might wish to obtain for each document a simple keyword describing its content, a set of keywords, a whole phrase, a paragraph, etc. On the other hand, the result of classification is always a document's assignment to one or more classes (in case of *multiclass classification*).

II.1 TDT project

The huge number of documents available on the Web is impossible to categorize by hand. In addition, some Web sites change quickly (such as those of news media) and to keep track of what they are speaking about, we are obliged to make use of tools for automatized *topic detection*. The *TDT project* (topic detection and tracking) has made available several corpora for testing accuracy of topic detection algorithms [3]. For instance, the TDT-2 Corpus contains approximately 60 000 news stories acquired from six major American media sites in the first half year of 1998. A hundred topics are manually identified for the corpus which is divided into training, development and testing parts. Moreover, the TDT initiative sets five principal goals for a hypothetical perfect TDT system to achieve:

- Story segmentation
- New event detection
- Topic detection
- Topic tracking
- Link detection

Story segmentation addresses the problem of capturing the moments when a news channel starts or stops telling a story (with a given topic). This task applies only to the audio portion of TDT corpus because text news are segmented already. *New event detection* deals with discovering that a new event, never known before, has just emerged. *Topic detection* decides over a story's topic (previously unknown), whereas *topic tracking* assigns each story to a specific topic (already known to the system). So it is very close to classification, indeed. Only, classification, under normal circumstances, is not limited by time and has the whole training corpus at its disposal. In *topic tracking* the decision must be made with only little time to look ahead (in the news feed time line). Likewise, topic detection is rather an incremental clustering for the same reasons. Finally, *link detection* is concerned with deciding whether two stories are linked, i.e. discussing the same event. Recently, *hierarchical topic detection*, which aims at creating a topics taxonomy, has seemed to replace *topic detection.*

II.1.1 State of the Art

[Flynn04] discusses the *topic detection* subtask of the TDT project and presents a system for clustering news reports. Moreover, he summarizes some conclusions taken from previous research work in this field. We will briefly state them here supposing that the reader of this report is familiar with the document vector space model and corresponding formulas.

Information retrieval (IR) systems often try to reduce document feature space excluding terms whose presence or absence in documents does not change their meaning. One example may be stop-words such as prepositions, articles, etc. Features are also selected according to their frequencies – *local*, *global* and *document frequency*. For example, many IR systems discard the terms that occur very rarely in very few documents. However, in the domain of news reports infrequent terms are often the most characteristic when talking about a particular event. Thus, Flynn et al. choose terms independently of their local frequency and for the global frequency, they throw away those terms that appear only once in the whole corpus.

Each selected term must be assigned a weight, which is mostly a product of a *local weight*, *global weight* and a *normalization factor*. Local and global weights are directly related to their respective frequencies whereas the normalization factor prevents long documents from having greater weights than short documents. Flynn experimented with combinations of local and global weights and he concludes that ANTF local weighting and IDF global weighting work best together (ANTF stands for augmented normalized term frequency). They do not say which normalization factor they use but assuming the standard cosine normalization, the weighting formula might look like this:

$$w_{ij} = [0.5 + 0.5(\frac{f_{ij}}{x_j})] \log \frac{N}{n_i} \frac{1}{\sqrt{\sum_{i=0}^{m} ([0.5 + 0.5(\frac{f_{ij}}{x_j}) \log \frac{N}{n_i}])^2}} \quad (3)$$

where w_{ij} is the weight of term i in document j, f_{ij} is the number of occurrences of term i in document j (term frequency), x_j is the greatest term frequency in document j, N is the number of documents in the corpus, n_i is the number of documents that have term i in them (document frequency), and m is the number of distinct terms taken into account when measuring similarity between documents (size of the feature vector). (In addition to classical information retrieval literature, [Chisholm99] presents a very nice overview of well-known and some new weighting formulas.)

The authors also discuss two main categories of clustering (partitioning and agglomeration) saying that partitioning approaches are generally faster but agglomerative techniques are more accurate. More specifically, they claim that hierarchical agglomerative clustering (HAC) with group average similarity appears to be the most effective in the TDT domain.

II.1.2 TopCat

TopCat (Topic Categories) by [Clifton04] is another interesting state-of-the-art project for identifying topics in news articles which makes use of the TDT corpus. Authors employ a hypergraph partitioning clustering mechanism on documents represented by sets of named entities – persons, organizations, and locations. Strictly said, they cluster their frequent itemsets, i.e. groups of names often occurring together. To obtain named entities they take advantage of a system called Alembic [Day97]. The reader is reminded that careful feature selection usually has a very little effect on results, while a bad selection can have a disastrous

outcome. Thus, choosing named entities is not straightforward. However, results show that TopCat is effective.

Alembic is a tool enabling quickly tagging large text corpora. Unfortunately, it is conceived for pre-tagging rather than tagging. So it supposes that a human expert will verify the tags in the end. Tags may be of a general nature, not just for labeling named entities as mentioned above. It works in a semi-automatic way, having an ergonomic user interface so as to minimize mouse clicks. It is able to derive a set of tagging rules on the basis of a training tagged corpus. For instance, it marks each proper noun after "Mr.", "Mrs.", "Dr." with a person label. The tagging rate is at most hundreds of tags per minute. The system itself is a set of various programs in different languages (including Lisp) and it runs under UNIX only. It is evident that it may spare some human effort in extracting personal names from a text corpus but it is not usable for processing very large collections fast, effectively and automatically. The question is also how to compose a rule for identifying a proper noun. If the first rule in the chain is badly defined, so are all others.

II.2 Linguistic Tools

Detecting topics in texts written in a natural language has to do with computational linguistics. Computational linguistics deals with computerized treatment of human language in written as well as spoken form (strictly said, some representation of spoken form). One important term in this context, which we borrow from "normal" linguistics, is a *concept*. A *concept* means a domain (field, area of human knowledge), an idea, or a class in object-oriented programming terminology. We may simply think of *concepts* as topics, thus finding *concepts* equals detecting topics. *Concepts* are often defined by typical *phrases* occurring in text. By phrases we mean sequences of words rather than syntactical sentences. Some authors like [Feldman98a] refer to *phrases* as *terms*, which is a little bit confusing with regard to information retrieval (IR) terminology where terms are lemmatized, i.e. words are stemmed to basic forms. We prefer the IR approach and we consider phrases consisting of terms. In general, finding phrases may be done in two ways: syntactically or statistically (simply looking for word co-occurrences). [Mitra97] showed that both methods had approximately the same effectiveness when employed for document indexing. For obvious reasons, computers use mostly statistic techniques.

II.2.1 Repeated Sequences

Repeated word sequences, sometimes referred to as *repeated segments*, phrases, co-occurrences, or collocations, are simply sequences of words that occur more than once in a text or corpus. Whether all words in the text are taken into account or some of them are omitted (via stoplists) is implementation dependent. More generally, we may think of symbols instead of words. Then, repeated sequences of symbols are patterns representing the original data from which they were obtained. In this general sense, repeated segments extraction and application reach far beyond the scope of computational linguistics and text mining. They are an important means of clustering, classification, topic detection and other machine learning and artificial intelligence techniques.

In the paragraphs below we will make the readers of this report familiar with three statistical approaches to repeated sequences extraction – a suffix tree based method, an inverted list based method, and a compression algorithm creating repeated sequences as a by-product. In [Tesar05] we presented our implementation and a comparison of both algorithms.

II.2.1.1 Suffix Tree

A *suffix tree* (ST hereafter) is a data structure that allows many problems on strings to be solved easily and quickly. It can be used to solve a large number of string issues that occur in text editing, searching and other application areas. The suffix tree originates in the 1970s (see [Weiner73]), but only in the 1990s it was modified for ST-phrases (repeated sequences) retrieval (see [Zamir98]). Since this method is natural language independent, there is a possibility to manipulate with documents in many languages simultaneously (see [Grolmus03a]). Other advantages are the document processing order independency and the possibility to regulate the length of retrieved phrases that is straight dependent on the adjusted suffix tree level. There exist many implementations (see [Sandeep04], [Ukkonen95] or [Andersson95]) suitable for various purposes. We had been inspired by [Zamir98], [Sandeep04], [Ukkonen95] and [Andersson95] when we implemented the procedure of creating a ST-structure we present below.

The general algorithm for the suffix tree structure construction convenient for discovering frequent phrases is similar to the algorithm depicted in [Zamir98]. In its simplest version, the suffix tree algorithm creates a tree structure that contains words in the order corresponding to their positions in the input documents. Each node of the suffix tree represents one word and the root represents the null word. Thus each path from the root represents a phrase containing the words labeling the nodes traversed. The suffix tree is a rooted and directed tree. For example, we want to create a suffix tree structure using the following sentences

"can drive trucks safely. men drive cars safely. men can drive trucks"

and we define the maximum tree height (thus the maximum phrases length) $m = 3$. The structure created in this way is shown in Figure II.1. When the suffix tree structure is created, the simplest way of obtaining single frequent phrases is to use a recursive procedure traversing the tree from the root node to leaf nodes. A non-recursive implementation is a relatively complex problem and there is no guarantee that we can achieve a better efficiency (while having the same complexity), because it depends on the current compiler implementation. The frequency of phrases is determined by the node that represents the last word in a particular phrase. From the set of nodes representing a particular phrase, this node is always situated at the bottom level of the suffix tree structure (when we consider the root as being at the top).

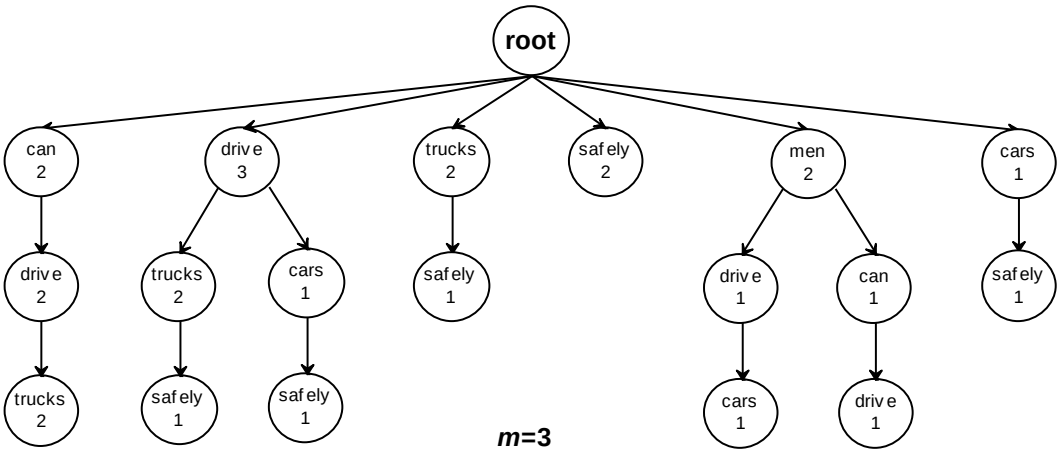

Figure II.1: Suffix tree structure example

A detailed description of the implemented ST-algorithm

The implemented suffix tree algorithm pseudo code is presented in Figure II.3. Time complexities for each partial task are shown as well. At the beginning, an input corpus is tokenized and stored to split list in such a manner that particular words are stored consecutively and sentences (or their parts bordered by punctuation etc.) are delimited by space (see Figure II.2). To do this, we used regular expressions which allow defining words and sentences delimiters. The split list is then used for the suffix tree structure construction. First of all, a root node has to be created. The root node is special, because it does not represent any word. Then a word from the split list‘s first position is read and added to a hash table of unique words in order to accelerate access to stored words.

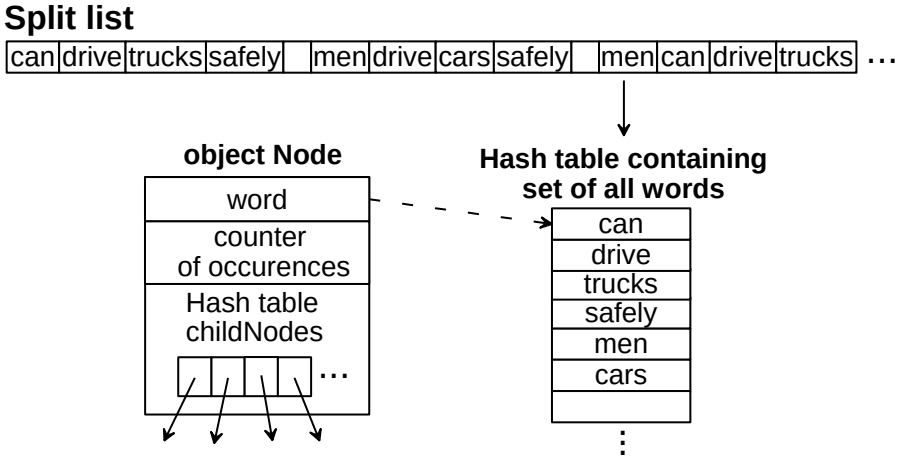

Figure II.2: Structure of a Node object

```
create a root node of ST-structure;                                          O(1)
while (input is not empty)
{
        clear parent list;                                                   O(N)
        while (get next input word != sentence delimiter)
        {
                add word to hash table;                                      O(N)
                add word node to root node;                                  O(N)
                add reference to parent list;                                O(N)

                for each node from parent list (except the last one added)
                {
                        add word node to current node;                       O(N*m)
                        add reference to parent list;                        O(N*m)
                        remove currently processed node from parent list;    O(N*m)
                }
        }
}

{
        traverse created ST-structure recursively to obtain ST-phrases       O(N)
}
```

Figure II.3: Pseudo code of suffix tree algorithm

Each of the suffix tree nodes represents just one input corpus word in form of a reference to the hash table, which is created in parallel. In this way we can considerably reduce the total memory requirements since single words at a suffix tree level deeper than one are repetitious. Nodes also contain information on how many times the phrase they represent occurs in the

input corpus and they provide a list of children *childNodes* (see Figure II.2). So if a word from the *split list* is read and added to the hash table, the root node has to be examined whether it contains a node that represents the current word because it is a parent node every time. If there is such a node, the counter of occurrences increases. Otherwise, a new node representing the current word is created and added to the list of children of the root node. In both cases, the node representing the current word is added to the end of the parent list.

Now we process the *parent list* which contains nodes added in the previous step (see Figure II.3). Because these nodes (except the last added node - it is ready for the next step) are parents of the current word (that means they represent the word that occurs before the current word in the input corpus), we can simply do the same we did in the case of the root node. The parent nodes processed are removed from the *parent list*. Naturally, a node representing the current word cannot be added to the *parent list* if its level is equal to the maximum suffix tree level. After that, a next word is read and the procedure described above repeats until the end of the *split list* is reached. Figure II.4 depicts the first four suffix tree construction steps using the sentence "can drive trucks safely" with the maximum level m set to three.

Complexity

As we can infer from Figure II.3, where N is the number of all words in the input corpus and m is the maximum suffix tree level, the time complexity is O(N + N*m + N). The space complexity of storing the tokenized text to memory is O(N), the hash table with unique words O(K), where K is number of unique words in the input corpus, all the suffix tree nodes O(N*m) in the worst case, and all the ST-phrases obtained O(N*m), in general. Thus, the total time and space complexities are both linear (see Figure II.8), which is conforming to [Zamir98].

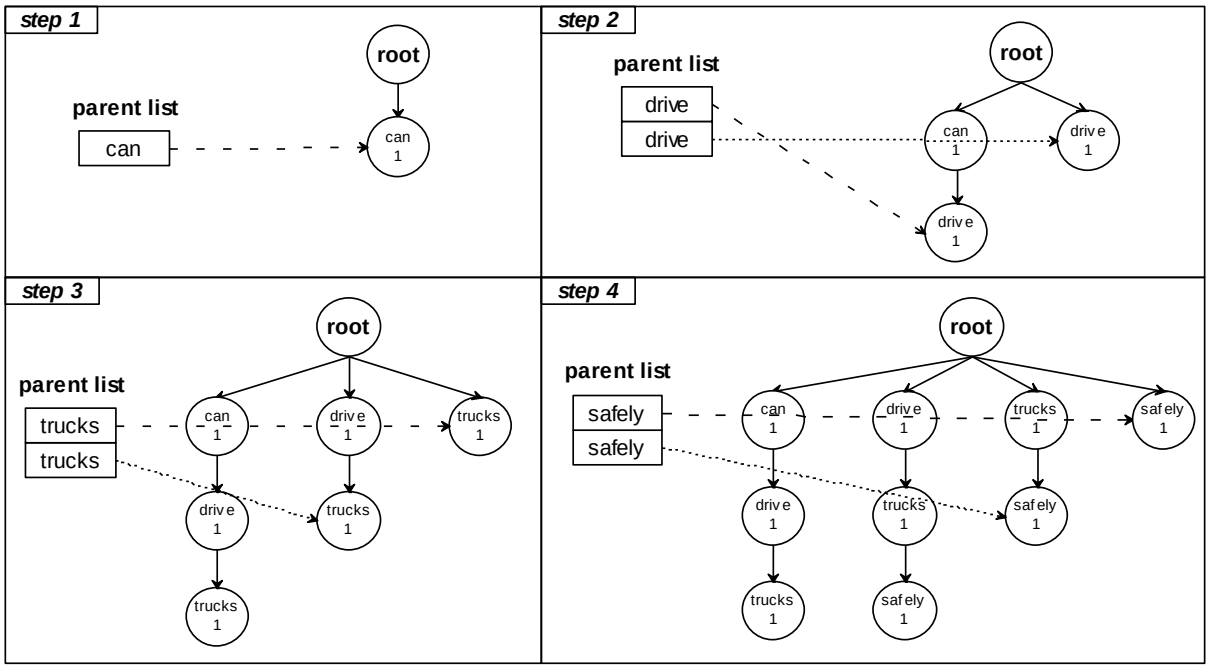

Figure II.4: Suffix tree construction example using *parent list* (*m* = 3)

II.2.1.2 Repeated Segments

Various authors used the term "repeated segments" in their papers among which we shall mention Justeson and Katz [Justeson95] and Lebart and Salem [Lebart94]. [Oueslati96]

employed repeated segments to acquire knowledge from a corpus. In those works, however, repeated segments had a more specific meaning and they were limited in definition by several constraints. They were thought of as *term candidates* for discovering new concepts and domains in text corpora. Thus, each term candidate had to contain a noun. Whether or not to include articles, prepositions, etc. in the term candidate was already application dependant. To determine the right morpho-syntactic patterns for generating an optimal number of term candidates is "*an open research question*" [Feldman98a]. See Table II.1 for an example of frequent repeated segments in English.

N N	candidate term
N PREP N	toys for kids
N N N	vector space model
ADJ N	little house
ADJ ADJ N	little red house
ADJ N N	little country house
N PREP ADJ N	room for pregnant women

Table II.1: Example of repeated segments

Of course, to detect nouns and exclude verbs, for instance, we need some external resources such as different filters, etc. Another problem with repeated segments employed for concepts discovery is that they do not work for all languages. For instance, the German language, which creates new words by simply concatenating existing words, doesn't enable finding repeated segments in the strict sense. The newly created word can occur no matter how many times in the corpus but, as it is a single word, it is never considered a concept.

A tool that was developed on the basis of Oueslati's work is called LIKES [4]. It is designed to allow for a number of linguistic tasks. To refine the discovery of term candidates (repeated segments) it makes use of a couple of filters such as a cutting filter with a list of words that cannot be part of a term candidate (mostly verbs and conjunctions), a grammatical filter with words that tend to introduce a segment (articles and pronouns) and so forth. These filters, of course, are language dependant and they must exist for each language in the corpus. LIKES itself is somewhat heavy weighted; it builds a tree of objects (paragraphs, sentences, words) for the whole corpus. However, it keeps track of segment contexts and relations. For instance, it is possible to find out where in the corpus (which text, paragraph and sentence) a particular segment is located. In the text below, repeated segments will mean pure repeated sequences without the limitations mentioned in the previous paragraph.

Algorithm

The algorithm pseudo code is presented in Figure II.5. Time complexities for each partial task are shown as well. Explanation will be given further. Our algorithm for discovering repeated segments does not have the necessity to take account of various filters because it considers each token in the corpus (here the tokens are words but they might be whatever symbols as well) as equal in terms of their morpho-syntactic meaning. Figure II.6 shows the most significant stages of the algorithm. We will follow the steps in the algorithm and give an example of its application. Consider the following input text (corpus):

Personal construct psychology. Personal construct psychology. Personal construct theory. Personal construct technology.

Pre-processing, i.e. tokenization, normalization, etc. of the input corpus is performed outside the algorithm, so the actual input is then an array of tokens (words). We create an inverted

index by adding each word to a hash table along with a list of its occurrences, i.e. with references to the corresponding places in the corpus where those words occur. Afterwards, with each word in the index we do the same operations. For instance, for the word personal, we look up its occurrences and create a list of words which follow this word at different places (see phase 1 in Figure II.5). The list will be sorted and only those words will remain that are part of sequences that occur two or more times. The other words are removed (see phase 2 in Figure II.5). Next, duplicate sequences are removed and the unique ones are counted in phase 3. Left subsequences of all sequences are added to the list in phase 4 and the same as in phase 3 is done in phase 5. Finally, the newly discovered segments are added to the list of repeated segments (phase 6). The whole process is repeated for another word in the inverted index. The sequence *construct psychology*, for example, will be found when processing the word construct. In theory, the maximum length of a repeated segment could be a half of that of the whole corpus. However, it is advisable to set it to a reasonable constant (in the example above the constant is three). Segments longer than this constant will be ignored.

```
// reads tokenized text and updates inverted index
foreach word in text
add to hash table;                    O(N)

foreach key in hash table             O(K)
{
        // creates list of words following each occurrence of key word
        get following words;          O(V)
        // modifies list so as only those sequences remain that occur twice at least
        reduce words list;            O(V + VlogV)
        // counts newly discovered segments and adds them
        remove empty columns;         O(V)
        remove duplicates;            O(V)
        add inclusions;               O(V + VlogV)
        add segments;                 O(V)
}
```

Figure II.5: Repeated segments – algorithm pseudo code

Complexity

Partial time complexities of individual operations are already noted in algorithm pseudo code in Figure II.5 where N is the number of words in the corpus, K is the number of unique words in the corpus (i.e. items in the hashtable) and V is the average number of occurrences of a word in the corpus. The evident time complexity of the whole algorithm in terms of these three variables is

$$O(N + K(V \log V + V)) \tag{4}$$

Obviously, V is N/K. Thus, substituting in (4) we get

$$O(N + K((N/K)\log(N/K) + N/K)) \tag{5}$$

and further

$$O(N + N + N\log(N/K)). \tag{6}$$

If K was N (i.e. each word in the corpus was different), the simplification to O(N) would be straightforward. But as K rather tends to be logN (see Figure II.7 and Zipf's law [Zipf49]), the total time complexity is supralinear; that means

$$O(N \log N). \tag{7}$$

If we focus our attention on the space complexity, we observe that keeping the tokenized text in memory is O(N), the hash table with occurrences, i.e. the inverted index, is O(N) (the number of all occurrences can never be larger than N) and the list of segments is O(N). Thus, for the overall space complexity we get O(N). We can conclude saying that the algorithm for discovering repeated segments presented above is supralinear in time and linear in space.

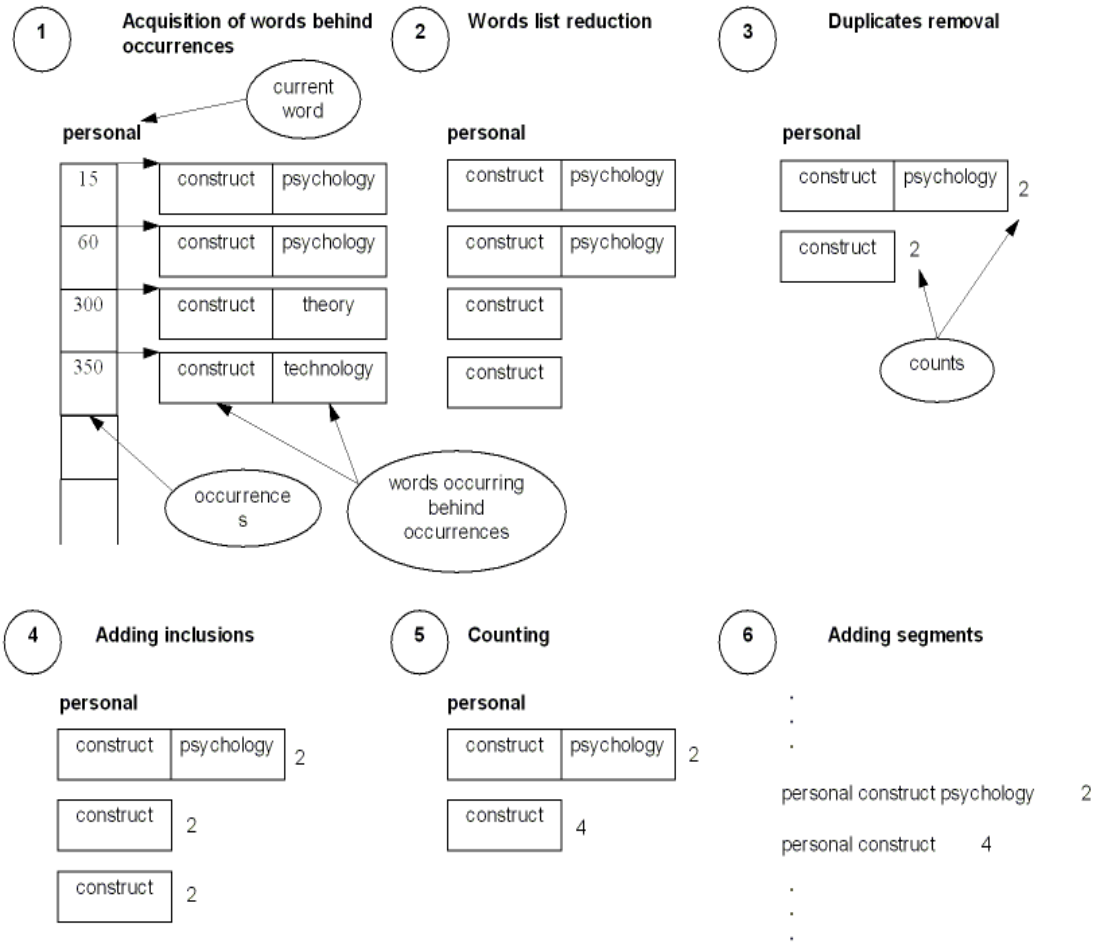

Figure II.6: Important phases of repeated segments discovery

II.2.1.3 Experiments

We ran each algorithm on five different corpora whose size ranged from 1 MB to 30 MB approximately, in terms of the total number of words up to several million words. We were interested in sequences of maximum lengths of three, four and five and in time and memory consumed by each algorithm. Both algorithms discovered all word sequences occurring in every single corpus, thus the sequences found were identical in either case. See Figure II.7 for the characteristics of used corpora. Charts in Figure II.8 confirm the theoretical assumptions we made about the algorithms' complexities. ST algorithm is linear in time as well as in space, whereas RS algorithm is supralinear in time and linear in space. However, the space consumption by ST algorithm is about twice as high as that by RS algorithm. (We implemented both algorithms in C#, compiled them on .NET Framework 1.1 and ran them on AMD Opteron 1.6 GHz with 2 GB RAM.)

We showed two algorithms for finding repeated sequences of words and there are many good reasons why we should be interested in those repeated sequences (and not always of words). For instance, they may be useful for classification and clustering. We might apply a standard method [Han00] but as a similarity of two objects we would use the ratio of common repeated sequences. Also, repeated sequences are usable for topic detection in sets of text, topic evolution and other text mining tasks (e.g. [Feldman98a]). Obtaining repeated sequences as a by-product of a compression algorithm called SEQUITUR is described in the next section. Possible applications include spam filtering [5], Web users profiles generation [Grolmus03b], compression [Grabowski99], document indexing [Hammouda04], statistical linguistics [Stubbs04] and others.

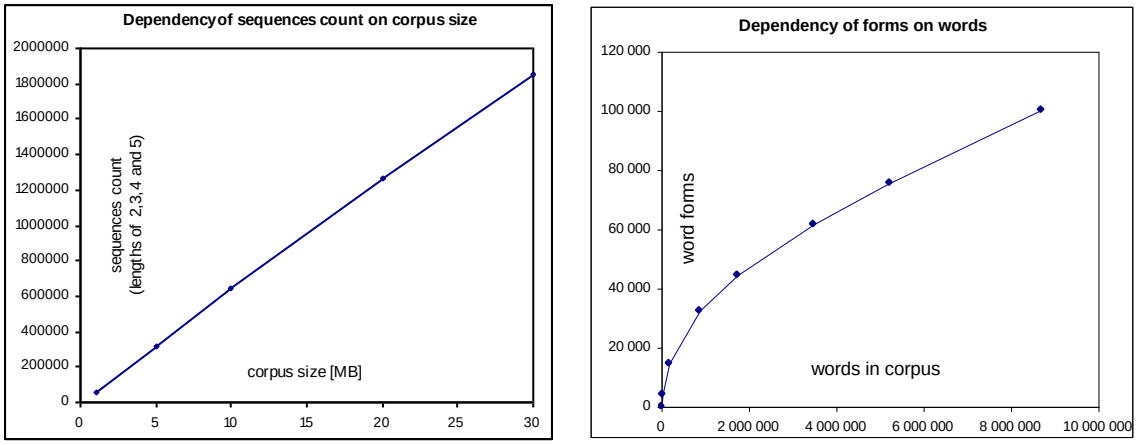

Figure II.7: Characteristics of used corpora

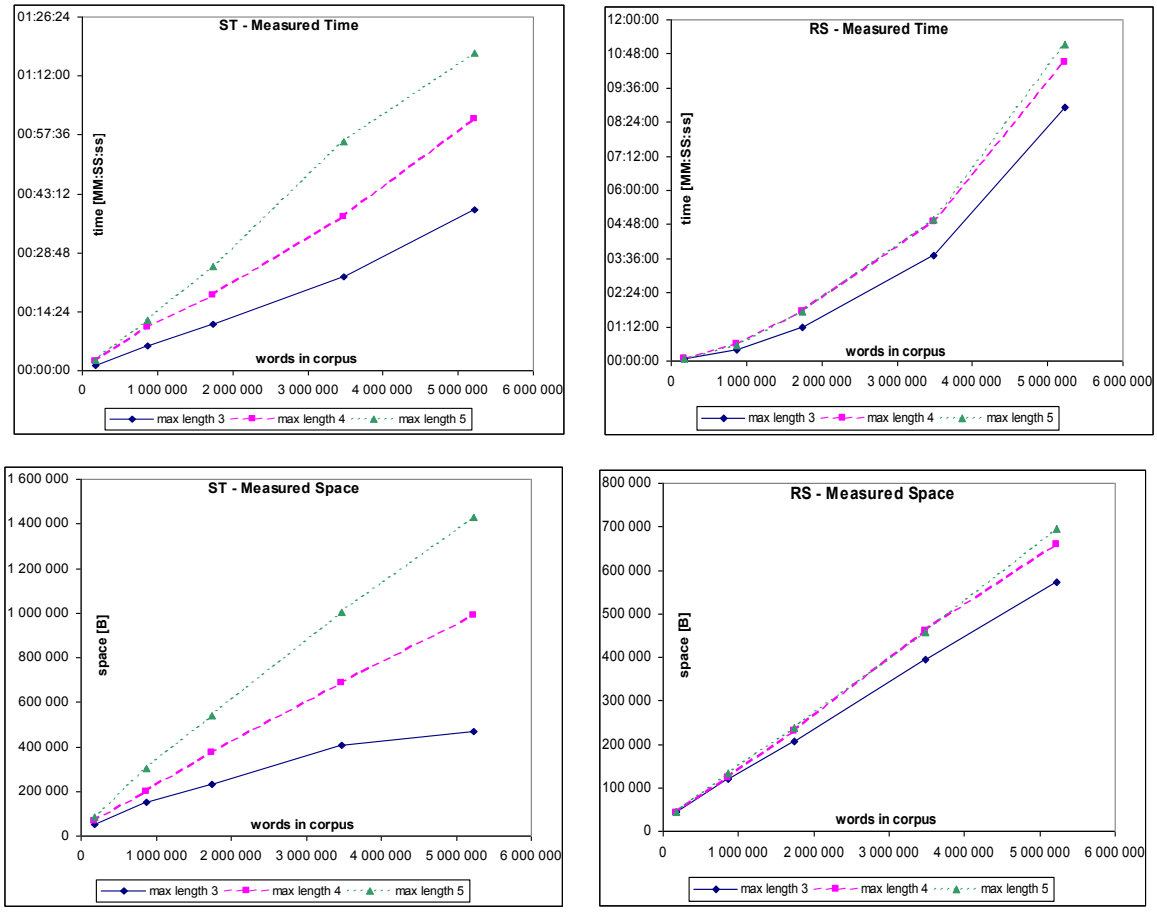

Figure II.8: Results achieved with both algorithms on sample corpora

II.2.1.4 SEQUITUR

SEQUITUR (see [Nevill-Manning97]) is a compression algorithm generating hierarchical grammars of the sequences being compressed. It can be stated concisely in the form of two constraints on a context-free grammar and it is linear with sequence length. The two constraints are:

- No pair of adjacent symbols appears more than once in the grammar (*digram uniqueness*)
- Every rule is used more than once (*rule utility*)

For example, both grammars in Table II.2 have satisfying properties:

Sequence	Grammar	Remark
abcdbc	$S \rightarrow aAdA$ $A \rightarrow bc$	Rule A is used more than once. All digrams aA, Ad, dA and bc occur only once.
abcdbcabcdbc	$S \rightarrow AA$ $A \rightarrow aBdB$ $B \rightarrow bc$	Both rules A and B are used more than once. All diagrams AA, aB, Bd, dB and bc occur only once.

Table II.2: Examples of grammars satisfying the constraints

SEQUITUR's operation consists of ensuring that both properties hold. The algorithm operates by enforcing the constraints on a grammar: when the *diagram uniqueness* constraint is violated, a new rule is formed, and when the *rule utility* constraint is violated, the useless rule is deleted. The fact that each rule (except the initial rule of the grammar) is repeated, allows us to consider each right-hand side as a repeated sequence of symbols. These symbols might be characters, words, notes, etc. according to SEQUITUR's application.

II.3 Taxonomies

A *taxonomy* is a hierarchy of classes. We are primarily concerned with topic classes. A topic taxonomy is a convenient means of searching for some particular information when we know (more or less) on which category of human knowledge we should focus our attention. The reader is certainly familiar with directory services offered by most Web search engines, the most popular of which are Google [6] and Yahoo! [7]. A *directory service* allows for a comfortable browsing in a hierarchy of categories when searching for a particular subject, as opposed to simple key word search when we do not know or do not want to specify the field of interest. More about search engines will be said in Chapter IV.

If we take a look at the taxonomy of Yahoo!, we see that there are twelve root topics. At Google there are fifteen of them. Ten topics are common - *Arts*, *Business*, *Computers*, *Health*, *News*, *Recreation*, *Reference*, *Regional*, *Science*, *Society*; five are unique to Google - *Games*, *Home*, *Kids and Teens*, *Shopping*, *Sports*; two are unique to Yahoo! - *Education*, *Entertainment*. (Both hierarchies may differ in regional versions.) We can see that the division at the root level is a little bit too fine with Google. *Games*, *Shopping* and *Sports* could be easily put together into an *Entertainment* category. Also, *Home* and *Kids and Teens* might be added to *Recreation* and *Society*, respectively. On the contrary, Yahoo's *Education* is a subtopic of *Society*.

But in general, both hierarchies are quite similar and they represent the current *ontology* of human knowledge. (We will discuss *ontologies* in Section II.1). More interestingly, they are both built manually, based on effort of many human editors. The *Open Directory Project* [40] behind Google Directories is open for public. In the next paragraphs we will try to answer the question why it is so difficult to create a topic taxonomy automatically.

II.3.1 Clustering

Constructing a taxonomy resembles hierarchical clustering. Let us recall that there are two ways of clustering a set of objects in a hierarchical manner – a partitioning and an agglomerative approach. In a partitioning method we begin with the whole object set in one cluster and we gradually break it down into more clusters with less objects by putting outliers into separate clusters until we end up with each object being on its own in a special cluster. These one-object clusters are called *singletons*. In hierarchical agglomerative clustering (HAC), on the contrary, we start from singletons, in each step we agglomerate (link, concatenate) the two most similar clusters (similarity measures as well as linkage criteria may vary) until we have all objects in one cluster. The clustering process can be visualized in the form of a tree referred to as a *dendrogram*. See [Han00] for an overview of clustering techniques and corresponding references. Figure II.9 is an example of HAC.

II.3.1.1 Incremental Clustering

Thus, given a set of documents on different topics, we could apply a HAC method to obtain a topic hierarchy and then use one of the topic detection techniques mentioned earlier to label the topics somehow. This seems to be straightforward until we realize that the initial document set can change dynamically. Imagine a Web crawler which is in charge of monitoring steadily a collection of Web sites. It downloads documents and builds a taxonomy upon them. The problem is that the Web sites collection will probably not remain the same all the time. Some documents may disappear, some others may be added, etc. To keep the taxonomy up to date it is necessary to reflect those modifications. [Hammouda04] calls this dynamic clustering an *incremental clustering* and he underlines several challenges:

- How to determine to which cluster the next object should be assigned? (a common issue for both incremental and non-incremental algorithms)
- How to deal with the problem of insertion order? (in other words: if object A precedes object B in input, will (and should) the clusters (and hierarchy) look the same as if B preceded A?)
- Once an object has been assigned to a cluster, should its assignment be frozen or is it allowed to be reassigned to other clusters later on? (i.e. should we prefer efficiency to accuracy?)

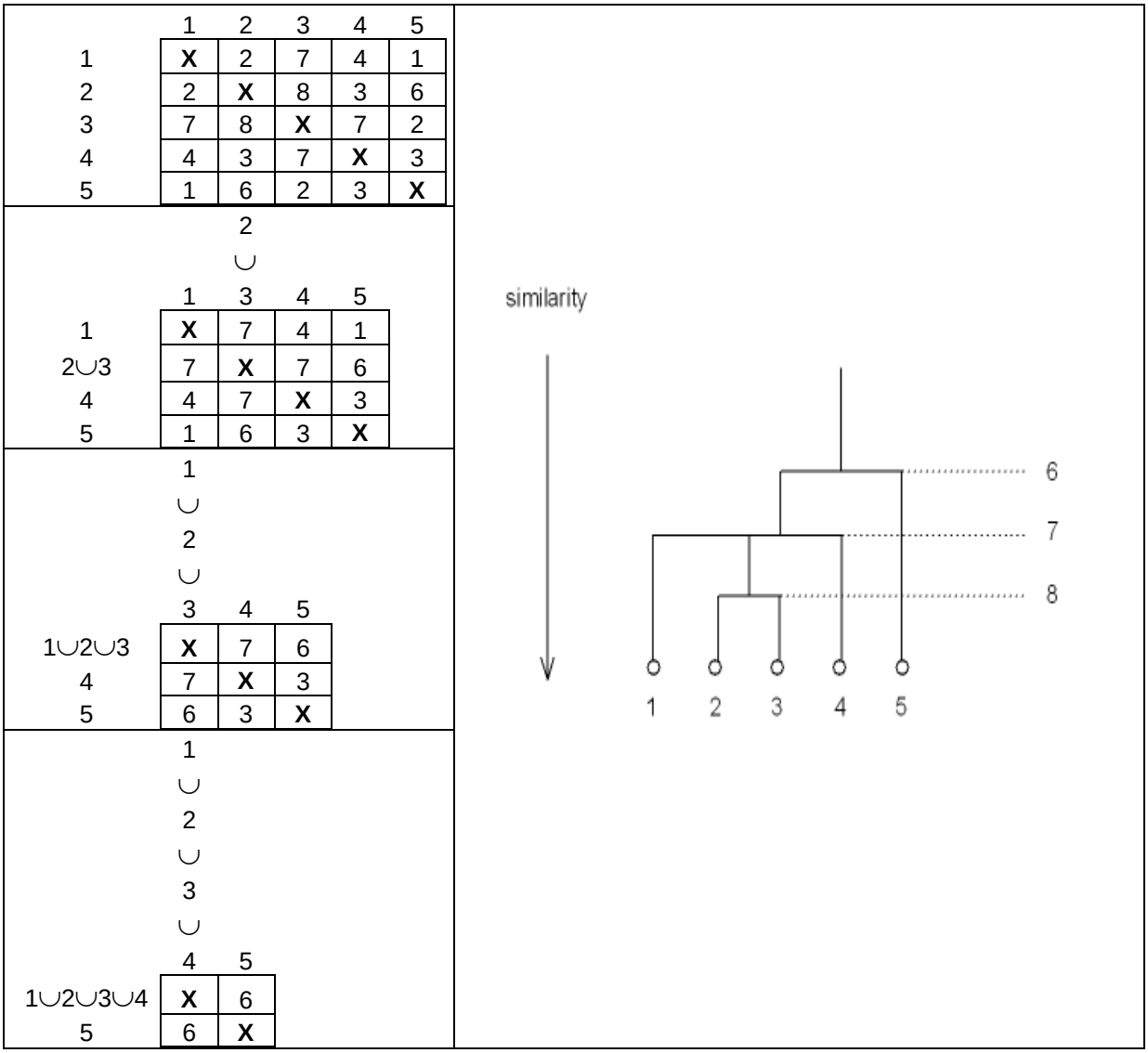

Figure II.9: Example of a hierarchical agglomerative clustering process and the corresponding dendrogram (values in matrices are intercluster similarities; *single link criterion* is used – the similarity between an existing cluster and a newly agglomerated cluster is chosen to be the highest similarity between the clusters in the new cluster and an existing cluster)

[Hammouda04] enumerates four basic incremental clustering approaches and then proposes a new method called *SHC – similarity histogram-based clustering*. A *cluster similarity histogram* is a tuple of intervals and a count for each interval. A perfect cluster would have only one interval (or a point corresponding to the highest possible similarity) and the associated count would be the number of all pair-wise document similarities in this cluster. On the contrary, a poor cluster would have all similarities in the minimum interval. The goal is to maximize the number of similarities in the high histogram levels of each cluster. So the algorithm receives a new document, and it calculates the similarity histogram for each existing cluster before and after a fictitious addition of the new document. If the similarity distribution (*histogram ratio*) of a cluster is enhanced or only slowly degraded, the document is added to that cluster. Otherwise it is not added. If a document is not added to any of the existing clusters, it is put to a newly formed cluster. The time complexity of SHC is $O(n^2)$. however, the authors claim that it is sub-quadratic in typical situations.

SHC is augmented with *reassignment strategy*. It keeps track of "bad" documents in clusters, i.e. documents that would increase their cluster's histogram ratio if they were removed, and it periodically checks whether they could be reassigned to another cluster while enhancing the histogram ratios of both the original and the recipient cluster. Note that SHC is not hierarchical. Constructing a hierarchy incrementally is a much more complex problem because reassigning a document might lead to a complete destruction of the current hierarchy in the worst case. There are many algorithms for incremental hierarchical clustering such as in [Nassar04], however, *it would be interesting to find out which perform best with dynamical updating of Web document taxonomies*.

II.3.2 Document Explorer

[Feldman98a] and [Feldman98b] presents a system called Document Explorer. This software allows for so called *text mining at the term level*, which means mining knowledge from texts regarded as collections of *terms* rather than single words. These terms would correspond to our *repeated segments* defined above. Document Explorer consists of several modules – document retrieval, term extraction, taxonomy creation and knowledge discovery. We will focus on the taxonomy creation module.

The user of Document Explorer is offered several tools for a semi-automatic construction of a terms taxonomy, mainly a *taxonomy editor* and a *taxonomy refiner*. An initial taxonomy is built automatically on the basis of meanings assigned to markup tags produced in the first phase (document retrieval). Then it is up to the user to refine the terms hierarchy. For instance, the user can drag and drop a part of the terms list as a new subtree in the taxonomy. He or she can define synonyms or group together similar terms by means of regular expressions. Also, the refiner analyzes the term frequent sets generated and it suggests to add them as siblings to the hierarchy. For example, if one the frequent sets is {bread, milk, sugar} and both "bread" and "milk" are present in the hierarchy under node "food", it proposes "sugar" to be placed under "food" as well.

Taxonomies enable users to specify mining tasks concisely and the authors remind that some text mining algorithms require term hierarchies: general association rules [Srikant95], maximal association rules and frequent maximal sets [Feldman97b].

II.4 Detecting Personal Web Pages

As an intersection of topic detection and Web surveillance we might be interested in finding personal Web pages, for instance, personal pages of researchers in a certain domain. Taking account of typical document trees on Web servers of various institutions, personal Web pages mostly occur a couple of levels below the root page. Let us consider a University Web site. The root page is the University home page pointing possibly to other organizational entities such as faculties. Each faculty home page could link to departments, departments to staff home pages and the like. The tree resembles an organization chart. But how can a Web robot recognize whether or not a Web page is personal?

[Shakes97] introduces a *metasearch engine* called Ahoy! meant to serve for finding personal homepages. Ahoy! makes use of *Dynamic Reference Sifting* (DRS) process in that it submits the name of the person to look for to several Web services, it filters their output and it suggests candidate home pages. It learns itself by extracting patterns from URLs of successfully found home pages and it uses this knowledge to generate URLs of home pages when a person's name, its institution and country are known. Even though Ahoy! did not analyze full texts of pages, a search took nine seconds on average! Unfortunately, Ahoy! was removed from service in 2000 and has not yet been replaced.

II.4.1 TyPWeb

TyPWeb [Beaudouin01] is an architecture that enables analyzing Web sites with regard to text, structure and links. Its authors employed it to study differences in content and structure between personal and commercial Web sites, among others. They investigated four corpora (two of them were personal Web sites, the other two were commercial) and they found out that it was possible to distinguish those two types of pages only on the basis of personal pronouns usage (i.e. frequency of occurrences). They defined six categories of personal pronouns: p1 = {I, my, myself}, p2 = {you, your, yourself}, p3 = {he, she, oneself}, p4 = {we, ourselves}, p5 = {you, yourselves}, p6 = {they, them, themselves}. (The original research was conducted in French – this is an approximate translation.) Table II.3 shows the resulting distribution of pronouns.

	total corpus (705 sites)	corpus 1 (239 sites)	corpus 2 (430 sites)	corpus 3 (22 sites)	corpus 4 (14 sites)
p1	24	26	23	13	14
p2	5	5	5	2	3
p3	34	35	34	25	33
p4	10	10	10	11	11
p5	21	18	20	44	34
p6	6	6	6	5	5
Σ	100	100	100	100	100

Table II.3: Distribution of pronouns (in %), from [Beaudouin01]

A fast glimpse at the table reveals that corpora 1 and 2 are personal sites, whereas corpora 3 and 4 are commercial sites. After neglecting sets p3 and p6 and recalculating the percentages the clear separation between personal and commercial Web sites is even more visible: corpus 1 has 63 % of *I/we* against 37 % of *you*, corpus 2 has the same relation 58 % to 42 %, while corpus 3 contains 66 % of *you* against 34 % of *I/we* and in corpus 4 it is 61 % to 39 %.

When we try to use a classical search engine such as Google to find a person's personal home page we often succeed to get it as the first among the URLs returned. However, regarding the HTML source of the page we quickly find out what is the source of success – that person's name is stated in the title or in the meta tags of the page. If a personal home page is not labeled this way it will very likely be put together with other (irrelevant) pages by the search engine. *The reflections above encourage us to create a kind of meta search engine for personal Web pages which would submit queries to Web services (like Ahoy!) and filter out personal home pages taking account of pronominal distributions (like TyPWeb).*

II.5 Ontologies

An *ontology* in the philosophical sense of the word is a science about being. It asks questions such as "What is existence?" and it steadily looks for objects that "exist" or do not "exist". In computer science, an *ontology* is a formal description of a (mostly hierarchical) structure of concepts in a particular domain and their relationships. The most obvious relationships are the "is a" and "has a" (sometimes also referred to as "part of") relationships, well known form the object oriented programming terminology. A simplified version of an ontology is a *taxonomy*, which takes into account only the "is a" relationship and whose structure is a tree. (Compare with section II.3.)

There are a number of formal languages for defining ontologies, in the Web domain mostly based on HTML, XML, or RDF, such as SHOE (Simple HTML Ontology Extensions),

DAML (DARPA Agent Markup Language), OIL (Ontology Inference Layer), etc. Usage of ontologies is particularly possible in knowledge representation, inductive reasoning, classification and so forth. There are computer systems centered on ontologies but their development is "not stable" [Richy02]. For instance, [Varlamis04] introduces a prototype application THESUS, which measures the similarity of Web pages by mapping extracted terms to ontology concepts. The same approach is employed to find out whether a user query matches a Web document. Useful links on ontologies may be found at Wikipedia [8].

II.5.1 KA2

KA2 is a Web page created by the group *Knowledge Annotation Initiative of the Knowledge Acquisition Community* lead by R. Benjamins [Benjamins98]. It represents a manually built ontology with an easy availability [9] and transparency of ten categories of knowledge acquisition, e.g. machine learning, ontologies, specification languages, knowledge management, etc. For each topic there is a description, approaches used, researchers, research groups, conferences, journals, Web pages and the like.

II.5.1.1 Experiments

The analysis made in early 2004 of the first level of the tree whose root is KA2 has shown that this page is not maintained. There are 41 invalid URLs out of 110 stated. The corpus acquired by downloading valid documents (and removing HTML tags) has 1.5 MB. It contains 126 906 words, 9 914 forms and 3 341 hapaxes (words with just one occurrence). Table II.4 is a list of the most frequent repeated segments (case sensitive) with stop words removed.

Rank	Frequency	Form
1	209	Data Mining
2	149	knowledge discovery
3	146	data mining
4	146	Knowledge Discovery
5	113	Knowledge Management
6	100	Knowledge Acquisition
7	97	Artificial Intelligence
8	90	Machine Learning
9	89	Knowledge Engineering
10	69	booktitle Proceedings
11	65	International Conference
12	62	Semantic Web
13	54	knowledge acquisition
14	54	knowledge representation
15	53	association rules
16	53	Information Systems
17	50	Knowledge discovery
18	49	Computer Science
19	48	Data mining
20	46	Association Rules

Table II.4: KA2 – most frequent repeated segments

Further, we applied the method of repeated segments to the data acquired by crawling the Web into the depth of five from the root of KA2. The crawl took 11 hrs 35 min and gathered over 45 000 documents in total. About 7 000 URLs were invalid and connection to almost 1 600 URLs timed out (timeout was set to 3 seconds). The crawl was performed by means of a breadth-first tree traversal. The gathered data had to be divided into ten categories as stated

in Table II.5. For each category a corpus consisting of documents in various (but reasonable) formats was obtained but only HTML-like documents (HTML, XML, SHTML…) have been taken into account. The table further shows the size of the original corpus and the reduced (HTML-like) corpus.

	Category	All docs	Size [MB]	HTML docs	Size [MB]
1	Machine learning	4 053	200	3 912	28
2	Reuse	2 719	223	2 234	8
3	Problem-solving methods	12 438	1 209	10 658	159
4	Ontologies	9 093	940	7 944	940
5	Specification languages	4 193	119	4 032	80
6	Validation & verification	69	0.7	69	0.7
7	Knowledge management	3 535	108	3 468	85
8	Knowledge acquisition methodologies	9 488	960	8 299	178
9	Evaluation of knowledge acquisition				
10	Knowledge elicitation	4	0.05	4	0.05
	Total	45 592	3 759	40 620	1 478

Table II.5: Corpora obtained by crawling KA2

II.5.1.2 Relations Among Topics

The corpus for category 9 is empty the cause of which we did not examine because we further worked with four of those categories only. One possible reason for that could be invalid links in the ontology. A problem arose when crawling the individual topics. Should URLs already visited in a previous category be crawled again in the current one? If we answer no, the more domains we have crawled until now, the less URLs remain for the future topics as the categories are often in overlay. This is in accordance with the real life because research areas are not disjoint. Therefore, we do not use a global hash table of URLs but local ones for each category.

The current implementation of the algorithm of repeated segments discovery (see Section II.2.1.2) in C# is very fast but limited by the main memory. A scalable algorithm suitable for very large corpora will have to be implemented in the future. For these reasons, the algorithm was applied to the topics 1, 2, 6 and 10 only. We must remark that the number and quality of repeated segments found depends a lot on the quality of language filters used. We adopted those in LIKES [4] for English and modified them a little bit. We observed that the more detailed the filters the less segments were discovered, which is the expected behaviour. If we added filters for other languages as well, the results would be a little different. Especially in topic 2 where a lot of French and Dutch texts appeared. Table II.6 summarizes what was obtained from our indexing engine – number of words, word forms, hapaxes, repeated segments (heads) and the process time for each topic (1 CPU Intel Pentium IV 2.7 GHz, 512 MB RAM, Win XP).

Topic	Words	Forms	Hapaxes	Segments	Time
1	3 746 606	94 267	42 740	170 959	2 min 44.5 sec
2	1 010 067	40 705	15 794	58 802	36.5 sec
6	93 918	3 253	1 064	2 401	2.8 sec
10	6 402	1 510	780	190	0.5 sec

Table II.6: Repeated segments of four topics

To what extent are the examined topics distinct? How close are they to each other? Can we construct a hierarchy from them? To answer these questions we computed the intersection

ratio of each pair of topics. We always compared 1 000 most frequent segments of one category to all segments of another category. Of course, topic 10 (K10) has less than 1 000 segments so all of its segments are considered. The results are shown in Table II.7. So, for instance, 17.2 % most frequent segments in K1 are contained in K2 as well whereas only 11.3 % most frequent segments in K2 are also in K1. The interpretation would be that K2 is a more general (broader) topic than K1.

Topic	**K1**	**K2**	**K6**	**K10**
K1	X	0.172	0.035	0.017
K2	0.113	X	0.010	0.003
K6	0.153	0.067	X	0.008
K10	0.363	0.232	0.121	X

Table II.7: Intersection ratios

We may notice that the values in rows as well as in columns have a decreasing trend, which is implied by the number of segments in the categories K1 – K10 being smaller and smaller. This property prevents us from saying categorically that there is no hierarchy in those four topics. A research task might be to find a mapping of the matrix onto the hierarchy and vice versa. (Attempts to do so have failed so far.) The advantage over standard clustering methods which use distances between term vectors is in that no such large and sparse vectors need to be created. Despite the fact that the sets of repeated segments of the four topics are not of about the same size, we may conclude from Table II.7 that the topics are at the same level in the taxonomy of knowledge acquisition and that the ontology KA2 is correct. By the way, according to [Varlamis04] "there has been very little research on similarity measures between sets of elements". *This may also indicate a research direction.*

III Web Links Analysis

As we stated in Section I.4, one domain of Web mining is concerned with exploring the topology of Web sites. The term topology is borrowed from graph theory and it means the structure of Web graph, in which the nodes are Web pages and the arcs are links pointing from one page to another. Obviously, it is a directed graph. Soon it was discovered that this structure could bear some information no less important than the actual contents of Web pages. For instance, researchers have noticed that Web pages can generally be divided into two categories: pages that link to many other pages and pages that are pointed to by many other pages. In fact, this behaviour resembles human society when we think of Web pages as humans.

III.1 Authorities and Hubs

Gibson, Kleinberg and others ([Gibson98], [Chakrabarti98]) explored the existence of *Web communities*. In doing so they introduced the notions of *authorities* and *hubs* and they developed a technique called HITS (Hyperlink-Induced Topic Search), which is based on them. The authors conducted a number of experiments with HITS or methods derived from HITS and they took a surprising conclusion that was in contrast to the common opinion that the World Wide Web was "becoming increasingly chaotic". A *hub* links to many pages, whereas an *authority* is linked to from many pages. Between these two entities there is a mutually reinforcing relationship – a good authority is linked to from many good hubs and a good hub links to many good authorities. A Web page can be a *hub* and *authority* at the same time. The following Figure III.1 shows an example of a Web community. The set S includes pages obtained with a query to a Web search engine, the extended set T contains all the pages linking to the pages in the set S and all the pages that are linked to from the pages in the set S. The size of the set S is limited by choosing only a certain number of results from the search engine.

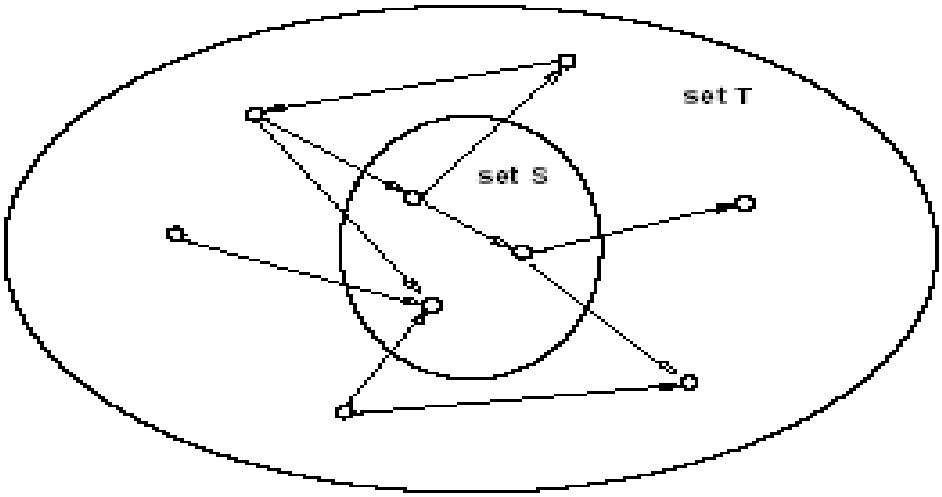

Figure III.1: Example of a Web community (from [Gibson98])

If we assign an *authority* weight a(p) and a *hub* weight h(p) to each page p, their values are computed as a sum of *hub* weights of the pages that link to it and as a sum of *authority* weights of the pages it links to, respectively. See equations (8) and (9).

$$a(p) = \sum_{q \to p} h(q) \tag{8}$$

$$h(p) = \sum_{p \to q} a(q) \tag{9}$$

At the beginning we initialize the values of all a(p) and h(p) to 1. We update them according to the formulae (8) and (9) in each iteration. We proceed like this with all the pages and we normalize the weights after each iteration. The authors prove that this iterative process converges to stable sets of weights of *authorities* and *hubs*. Then say ten greatest authorities and ten greatest hubs can be denoted as the *core of a community*.

Previous thoughts can be expressed in the matrix notation. Let $\mathbf{A} = [a_{ij}]$ be the adjacency matrix of a directed Web graph (similar to that in Figure III.1), where $a_{ij} = 1$ when the page i links to page j and 0 otherwise. Let **h** and **a** be vectors corresponding to the weights of hubs and authorities of all pages. We will repeatedly perform the following operations:

$$\mathbf{h} \leftarrow \mathbf{A}\,\mathbf{a} \qquad\qquad \mathbf{a} \leftarrow \mathbf{A}^T\,\mathbf{h}$$

From the classical matrix theory it implies that with an appropriate renormalization **h** (respectively **a**) converges to the principal eigenvector $\mathbf{A}\mathbf{A}^T$ (respectively $\mathbf{A}^T\mathbf{A}$). Kleinberg shows further that the non-principal eigenvectors of these matrices correspond to the authorities and hubs of the "non-principal" Web communities.

III.2 Extended Authorities and Hubs

We can also apply the idea of *hubs* and *authorities* to graphs with other kinds of nodes. In the previous case each node is a Web page but it could be a researcher, a research group or an institution as well. If a node is a researcher the arcs coming to this node are citations of this scientist (strictly said citations of his/her publications) made by other researchers (in their publications). On the other hand, the arcs pointing from this node to the others are citations of the other researchers. Of course, we might group researchers according to co-authorship, membership to institutions and so on. So the citations here are not meant to be references (links) from Web pages but directly from papers (publications). In the case of Web citations the cited entity can be easily recognized by its URL. It is more complicated with paper citations – it is necessary to find the references section in the paper, to retrieve the individual citations in it and possibly to determine the cited object. Here we must work with a certain ambiguity because not all citations have the same format, not all authors are stated with their second given names and so on. At the beginning we can let coexist the obtained values of Web and citation (paper) authorities. *We can find out whether there is some correlation between them and a "universal authority" may be derived from them later, saying which author is frequently cited both on the Web and in publications*.

IV Web Search Engines

In February 1999, Lawrence and Giles estimated the number of publicly accessible Web servers to be 2.8 million and the number of Web pages about 800 million [Lawrence99a]. The method used was random sampling of IP addresses, then removing servers behind firewalls, requiring authorization or containing no content. Then 2 500 random servers were crawled trying to get all pages on them. The resulting number of pages derived from the average number of pages on the random servers and their estimate of all servers. They also found out that there was 6 TB (terabytes) of textual content on the Web and that 6 % of the pages were scientific or educational. According to their research none of the search engines covered more than 16 % of the Web. The total number of Web servers estimated by Netcraft [10] was 62 million approx. in April 2005.

IV.1 Web Scope

With the information above we can make an estimate of the current Web size. Provided the relation between Web servers and pages is the same as in [Lawrence99a] there would be about 17.7 billion Web pages at present ($800 / 2.8 \approx 17\,700 / 62$). If we assume that Google's 8 billion Web pages (see Section IV.2.1) cover 16 % of the Web, there would be 50 billion documents on the Web. Thus, we can guess that there are **18 – 50 billion Web documents** ($18 \times 10^9 - 50 \times 10^9$). Again, retaining the relations from [Lawrence99a] that would mean 135 – 375 TB of textual information. The numbers corresponding to scientific and educational pages are then 1 080 - 3 000 million Web pages and 8 – 23 TB of textual content. To find some information in such an enormous amount of documents it is evident that the need for a Web search engine is vital.

IV.2 Searching the Web

Useful references regarding Web search are [11], [12], [13] and [Fiala03]. The Web growth is exponential and bibliographic control does not exist. To find a particular piece of information the use of a search engine is a necessity. The term *search engine* is rather universal; a finer division may be used:

- *search engines*
- *directory services*
- *metasearch engines*
- *search providers*

In this grouping *search engines* such as Google, AltaVista, Excite, HotBot, Lycos do keyword searches against a database, *directory services* (Yahoo!, LookSmart, Britannica, The Open Directory) sometimes also called *subject guides* offer information sorted in subject directories, *metasearch engines* (also *metacrawlers*) such as Dogpile, Mamma, Metacrawler or SawySearch send search requests to several *search engines* and *search providers* provide the underlying database and search engine for external partners. Examples of *search providers* might be Inktomi and Fast Search.

Directory services are fine for browsing general topics but for finding specific information *search engines* are more useful. Various factors influence the results from each. In particular, the size of their database, frequency of its update, search capability and design (algorithms) resulting in different speed have to be taken into account. The best use of *metasearch engines*

is submitting queries on obscure items to find out if something can be found on the Web at all. One of the current Inktomi partners is MSN Web Search.

IV.2.1 Google

Due to Google's superiority to other *search engines* in almost all features we are going to take a closer look at it as a representative of this category. It launched officially on September 21, 1999 with Alpha and Beta test versions released earlier. Since then it has pushed through with its relevance linking based on pages link analysis (the patented *PageRank* method), cached (archived) pages and a rapid growth. In June 2000 it announced a database of over 560 million pages and they moved up their claim up to 3 billion by November 2002. As of April 12, 2005 the number is 8 058 044 651 Web pages [14].

Googlebot is Google's Web-crawling robot written in C++ programming language. It collects documents from the Web to build a searchable index for the Google search engine. It follows HREF and SRC links on the Web pages. It is registered with the Web robots database [16] and it obeys the *Robot Exclusion Standard* [17] so it is possible to prevent it from visiting a site or a particular URL.

IV.2.1.1 Databases

The main Google database consists of four parts:

- *indexed Web pages*
- *daily reindexed Web pages*
- *unindexed URLs*
- *other file types*

Indexed Web pages are Web pages whose words have been indexed, i.e. some records have been made about what terms and how many times they occur on a specific page. Typically, the terms are sorted descending as in an inverted index. *Daily reindexed Web pages* are the same, except that Google reindexes them "every day". These pages display the date they were last refreshed after the URL and size in Google's results. *Unindexed URLs* represent URLs for Web pages or documents that Google's spider (Googlebot) has not actually visited and has not indexed. *Other file types* are Web-accessible documents that are not HTML-like Web pages, such as Adobe Acrobat PDF (.pdf), PostScript (.ps), Microsoft Word (.doc), Excel (.xls), PowerPoint (.ppt), Rich Text Format (.rtf) and others.

IV.2.1.2 Comparison With Other Search Engines

Various studies [17], [18] have shown that Google outperforms other search engines in terms of the size of its databases, frequency of Web crawls (i.e. "freshness" of documents in the repository), rapidity of responses to user queries, richness of its databases (as to the number of various document formats), and so on. Exact numbers may be found in [Fiala03]. We do not present them here because they usually change very quickly. But we do present a summary of Google's properties in Tables IV.1 and IV.2. Google has recently introduced many services related to Web searching such as *Google Scholar* (for finding bibliographic citations), *Google Local* (for finding goods and services near users' whereabouts), *Froogle* (for finding the cheapest products on the Web), and particularly *Google Desktop*, which brings the power of Google's indexing capabilities to users' personal computers. All in all, nothing seems to threaten its position in the near future.

Strength	Description
size	It has the largest database including many file types.
relevance	Pages linked from others with a greater weight given to authoritative sites are ranked higher (PageRank analysis).
cached archive	It is the only search engine providing access to pages at the time they were indexed.
freshness	The average time of page reindexing is one month.
special query terms	It offers a number of special query terms enabling very specific searches.

Table IV.1: Google's strengths

Weakness	Description
limited Boolean search	The full Boolean searching with the ability to nest operators is not supported.
case insensitivity	No case-sensitive search is possible.
no truncation	There is no automatic plural or singular searching or search for words with the same word root. There is only one way of using wildcards such as "*" in the phrase "pay attention * fire" where it matches any word in that position. (Very useful in finding the correct English prepositions.)
no similarity	Google searches for exact, not for similar words.
size limited indexing	Of course, it indexes only some parts (e.g. some first 100 kB) of very large documents.

Table IV.2: Google's weaknesses

IV.2.2 Specialized Search Engines

There are many specialized search engines that could be particularly used in the support of researchers. We will name only a few of them: OJOSE [19], Scirus [20] or Phibot [21]. OJOSE is a metasearch engine, which submits queries to scientific databases, online journals or other search engines. Scirus and Phibot index research Web sites and electronic journals. Phibot offers improved relevance algorithms and indexes some of the Web sites in 15-minute intervals! ISI Web of Science [22] - ISI stands for Institute for Scientific Information - enables users to search a database consisting primarily of papers from about 8 500 research journals. In addition to journals, specific Web sites are also included in the database. See [23] and [24] for information on how the journals and Web sites are selected. There are weekly updates, with items usually appearing 3 to 8 weeks after publication [25]. ISI Web of Science is a **commercial product**. On the contrary, ResearchIndex (formerly CiteSeer) [26] is free and we will discuss it in Section V.2.

IV.3 Architecture of Google

Information in this Section is taken primarily from [Brin98]. In 1998, Google was an academic research project, therefore its internals were communicated to the public. Since

then, Google has gone a long way from there: it is a commercial product and the know-how is, naturally, not transmitted further by its authors. However, we may assume that the core of its implementation has remained the same.

IV.3.1 Google Features

Google seems to obey the motto "high precision is important even at the expense of recall". It uses a Web page ranking method called PageRank to present the most relevant results upon a user query. PageRank is based on the link structure of the Web. It is a more complex version of citation ranking of scientific literature and it is discussed in more detail in Section IV.4. Apart from PageRank, Google has several other innovative features:

- anchor text (unlike other search engines, it associates the text within links with the pages they point to)
- proximity search (in multi-word searches pages with the words occurring close to each other are ranked higher)
- word presentation (it takes into account details such as font, HTML element in which the word occurs, etc.)
- pages repository (full HTML sources of Web pages are available in a repository)

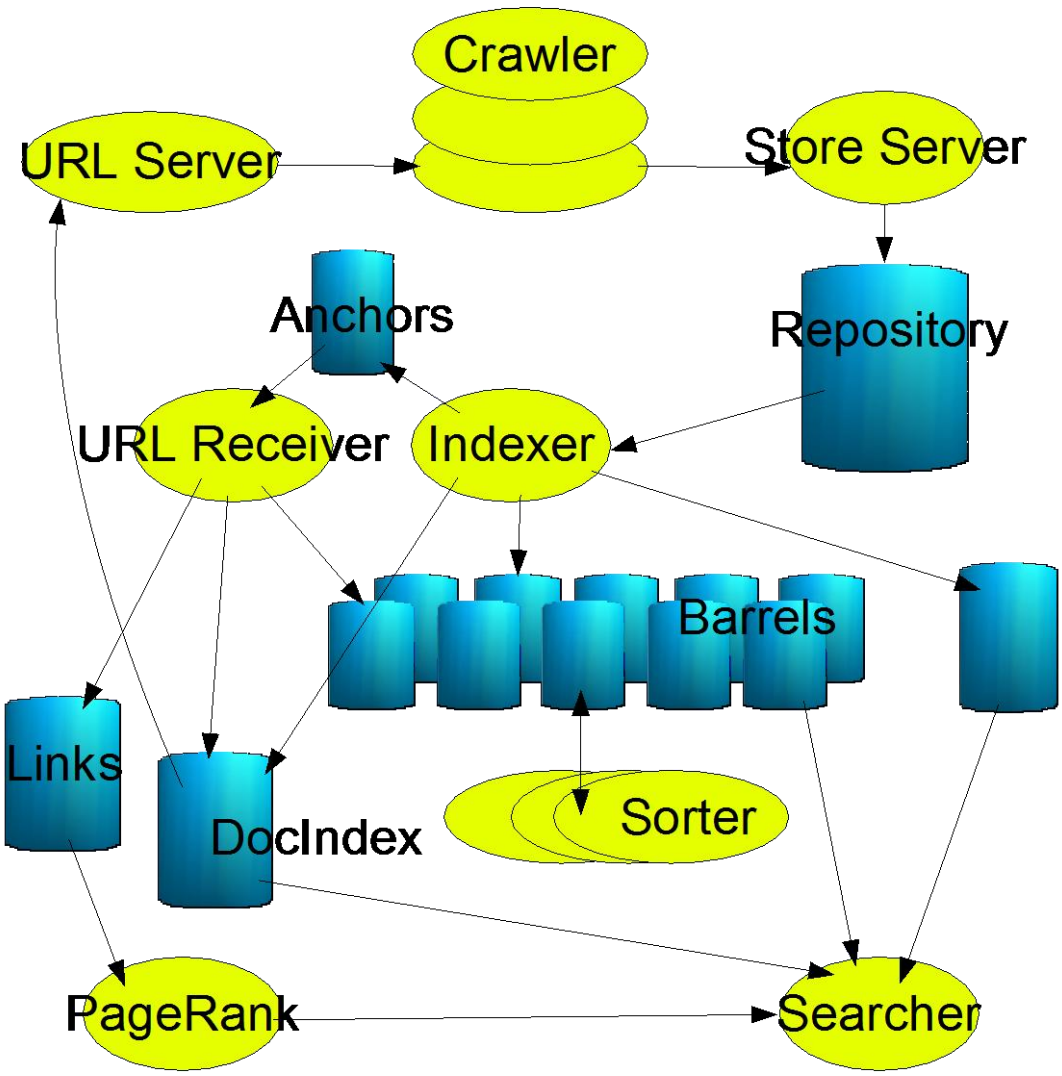

Figure IV.1: Google architecture (from [Brin98])

IV.3.2 Google Anatomy

The high level Google architecture is depicted in Figure IV.1. Several *crawlers* download Web pages in parallel. What pages to download is told them by the *URL server*. The *store server* receives the downloaded Web documents, it compresses them and it places them to the *repository*. The actual indexing of documents is done by the *indexer* and the *sorter*. The *indexer* uncompresses documents in the repository and parses them. Each document is converted into a set of word occurrences and other information on these words called hits. Hits are put to the barrels so as to create a forward index (document → terms) sorted by docID. A docID is a unique identifier associated with each document generated whenever a new document URL is found on a Web page. (We use the terms page and document arbitrarily.) When parsing the documents, the *indexer* adds information on all links to the *anchors* file. (The word anchor refers to the <A> tag in HTML including its HREF attribute pointing to another place. Evidently, the <IMG> tag with the SRC attribute is also a source of links.) Later on, it is possible to find out from the *anchors* file where each link points from and to, and also the link text (i.e. the text between the <A> and </A> tags).

The *URL Resolver* takes the links in the *anchors* file and transforms relative URLs to absolute URLs. For instance, it takes the link image1.gif and it reads that the page linking to it has the URL http://www.ulp.fr/index.html, so it composes the absolute URL of the image as http://www.ulp.fr/image1.gif. This absolute URL is then associated with its docID (in *docIndex*). Anchor texts are put to the index in *barrels* as if they occurred on the page they point to. The *URL Resolver* generates a *links* database as well, in which there are pairs of docIDs. This database will be used to compute PageRank values for all pages. The *sorter* sorts the barrels by words (by wordID) thus creating an inverted index (term → documents). The barrels have been sorted by docID up to now. Several sorters run in parallel. A list of wordIDs and offsets into the inverted index along with a *lexicon* produced by the *indexer* is employed to build a new *lexicon* utilized by the *searcher*. The *searcher* is run by a Web server [14]. It makes use of the lexicon, along with the inverted index and PageRanks to answer user queries (see Figure IV.2).

1. Parse the query.
2. Convert query words into wordIDs.
3. Seek to the start of the doclist in the short barrel for every word in the query.
4. Scan through the doclists until there is a document matching all the search terms.
5. Compute the rank of that document for the query.
6. If we are in the short barrels and at the end of any doclist, seek to the start of the doclist in the full barrel for every query word and go to step 4.
7. If we are not at the end of any doclist go to step 4.
8. Sort the documents that have matched by rank and return the top k.

Figure IV.2: Google query evaluation

A few remarks to Figure IV.2: A doclist is a list of docIDs with hit lists. It represents all occurrences of a particular word in all documents. Note that there are two sets of inverted barrels – one set for title and anchor hits (short barrels) and the other for full text hits (full barrels). This way, the short barrels are checked first and if there are not enough matches within them the full barrels are scanned. The final rank of a document is a combination of its PageRank and a score that takes account of the type of hits (title, anchor, URL, common text...) and the proximity of hits in multi-word queries (documents with search terms close to each other are ranked higher).

IV.3.3 Notes

We will omit details on data structures used in Google. Instead, we will recall a couple of noteworthy facts. Google is implemented in C and C++ and some parts of it are written in Python (URL resolver and crawler). It is designed to avoid disk seeks whenever possible because a disk seek takes about 10 ms on average. To satisfy its storage needs it takes advantage of virtual files spanning across multiple file systems. Documents in the repository are compressed using zlib [27]. All in all, the authors of Google insist that "a Web search engine is a very rich environment for research ideas" [Brin98].

IV.4 PageRank and Other Web Page Ranking Schemes

We will first introduce an intuitive (simplified) version of PageRank from [Page98] and then we will enhance it within a general formalistic framework for ranking methods by [Diligenti04] in Section IV.4.2.

IV.4.1 Intuitive PageRank

A clear analogy to PageRank is the calculation of impact of research publications in terms of the number of citations they have. In the Web domain a page has a citation when there is another Web page linking to it. From the point of view of the cited page this link is called a *backlink*. PageRank does something more than simply counting backlinks in that it assigns different weights to backlinks. As a result, a Web page can have a high rank if it has many backlinks (citations) as well as if it has only few but highly rated backlinks.

Let u be a Web page and B_u the set of pages that point to u. Then let v be a Web page and N_v the number of links from v. Finally, let c be a normalization factor for the total rank of all Web pages to be constant. Then the rank (simplified PageRank) R of u is computed like this:

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v} \tag{10}$$

Note that the formula (10) is recursive – for computing any page rank we need to know the others. Therefore, we must start with a vector of ranks initialized to some (arbitrary) values, iteratively update those ranks using (10) and wait until they converge. (The full version of PageRank converged after about 50 iterations according to [Page98].) Figure IV.3 is an example of a page rank calculation. There is a little problem with so called *rank sinks*. Those are closed loops of pages that accumulate rank but never distribute it further, such as in Figure IV.4. To overcome this trouble some modifications of the ranking function must be made which will be explained in the formal definition in Section IV.4.2.

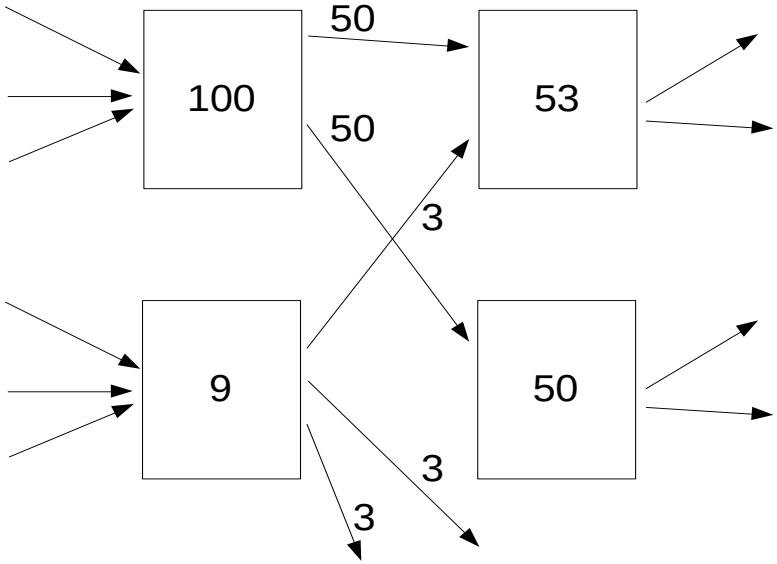

Figure IV.3: Simplified PageRank calculation

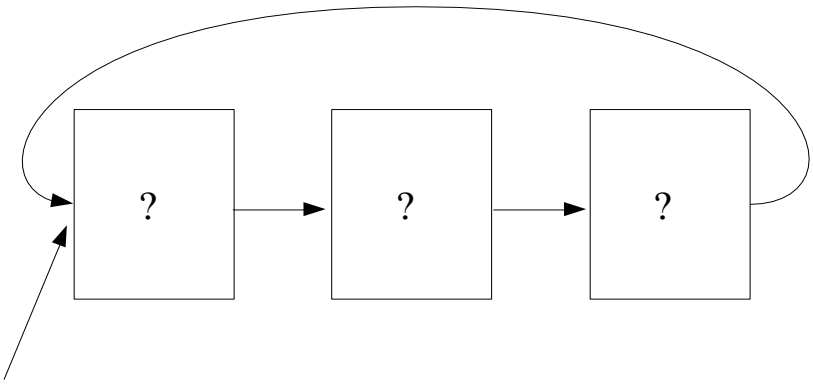

Figure IV.4: Rank sink

IV.4.1.1 PageRank Usage

The authors suggest a usage of PageRank in Web search, personalized Web search, browsing, estimating Web traffic and others. The idea in Web browsing is to display in a Web browser the rank of the corresponding page next to each link. This would enable a user to click only on links to highly ranked pages. Apparently, that requires having a proxy server between the browser and Web servers that communicates with Google. Google Toolbar [28] offers the PageRank value of the page currently being displayed in the browser.

IV.4.2 Formal Approach

[Diligenti04] distinguishes *horizontal* and *vertical ranking* methods. Horizontal rankings are only based on the Web graph topology and do not take into account the contents of Web pages. PageRank and HITS (see Section III.1) are both horizontal. Vertical (focused) rankings are useful for topic search. Diligenti's probabilistic framework is based on *random walks* in that the relevance (rank) x_p of a page p is computed as the probability of visiting that page in a random walk on the Web graph. The most popular pages (i.e. most often cited) are the most likely to be visited during a random walk.

IV.4.2.1 Random Walk

A random walk in the context of Web browsing is a mathematical model of actions taken by a generic Web surfer. At each step of the walk, the surfer can perform one of the following

actions: jump to any Web page (action *j*), follow a link to another page (action *l*), follow a backlink from the current page (action *b*), stay where he or she is (action *s*). Thus, the set of atomic actions is $O = \{j, l, b, s\}$. At each step, the behaviour of the surfer depends on the current page. If he finds it interesting, he will probably click on a link there. If he finds it boring, he types another URL to the address bar of his Web browser. So the surfer's behaviour can be modeled by a set of conditional probabilities depending on the current page *q*:

- $x(p|q, l)$: probability of following a link from page *q* to page *p*
- $x(p|q, b)$: probability of following a backlink from *q* to *p*
- $x(p|q, j)$: probability of jumping from *q* to *p*
- $x(b|q)$: probability of staying on *q*

If *G* is a Web graph, *p* and q are Web pages ($p,q \in G$), *ch(q)* is the set of pages pointed to by *q* (children of *q*), and *pa(q)* are the pages linking to *q* (parents of *q*) then the following constraints have to be satisfied for each page *q*:

$$\sum_{p\in G} x(p \mid q, j) = 1, \qquad \sum_{p\in ch(q)} x(p \mid q, l) = 1, \qquad \sum_{p\in pa(q)} x(p \mid q, b) = 1 \tag{11}$$

Evidently, the first constraint in (11) includes the case of remaining on the page because p can be the same as *q*. The probabilistic random walk model can be made use of to compute the probability $x_p(t)$ – that the surfer is located on page *p* in time *t*. The probability distribution on all pages is represented by the vector $\boldsymbol{x}(t) = [x_1(t),\ldots,x_N(t)]'$ where N is the total number of pages. The probabilities $x_p(t)$ are updated in each step of the random walk according to the following formula:

$$\begin{aligned} x_p(t+1) &= \sum_{q\in G} x(p \mid q) x_q(t) = \\ &= \sum_{q\in G} x(p \mid q, j) x(j \mid q) x_q(t) + \sum_{q\in pa(p)} x(p \mid q, l) x(l \mid q) x_q(t) + \\ &+ \sum_{q\in ch(p)} x(p \mid q, b) x(b \mid q) x_q(t) + x(s \mid p) x_p(t) \end{aligned} \tag{12}$$

where the probability $x(p|q)$ of moving from page *q* to page *p* is expanded considering the user's actions. The probabilities $x(j|q)$, $x(l|q)$, and $x(b|q)$ are general probabilities of jumping to a random page from page *q*, following a link from *q*, and following a backlink from *q*, respectively, without specifying a target page.

Now we will move to a matrix notation. The probabilities defining the random surfer model may be organized in a couple of N × N matrices:

- *forward matrix* Δ whose element (p,q) is the probability x(p|q,l)
- *backward matrix* Γ of the probabilities x(p|q,b)
- *jump matrix* Σ gathering the probabilities x(p|q,j)

We can also define a set of *action* matrices that inform about the probabilities of the individual actions taken on each page *q*. These matrices are N × N diagonal matrices having $x(j|q)$, $x(l|q)$, $x(b|q)$ and $x(s|q)$ as their diagonal values (q,q). We will denote those matrices $\boldsymbol{D}_j$, $\boldsymbol{D}_l$, $\boldsymbol{D}_b$ and $\boldsymbol{D}_s$, respectively. We can then restate (12) as

$$\boldsymbol{x}(\mathrm{t}+1) = (\Sigma \cdot \boldsymbol{D}_j)'\boldsymbol{x}(\mathrm{t}) + (\Delta \cdot \boldsymbol{D}_l)'\boldsymbol{x}(\mathrm{t}) + (\Gamma \cdot \boldsymbol{D}_b)'\boldsymbol{x}(\mathrm{t}) + (\boldsymbol{D}_s)'\boldsymbol{x}(\mathrm{t}) \tag{13}$$

By defining the transition matrix as

$$\boldsymbol{T} = (\Sigma \cdot \boldsymbol{D}_j + \Delta \cdot \boldsymbol{D}_l + \Gamma \cdot \boldsymbol{D}_b + \boldsymbol{D}_s)'$$

we can write (13) in the following way:

$$\boldsymbol{x}(\mathrm{t} + 1) = \boldsymbol{T} \cdot \mathbf{x}(\mathrm{t}). \tag{14}$$

From the initial distribution of probabilities $\boldsymbol{x}(0)$ we can compute a distribution in any time step *t*:

$$\boldsymbol{x}(\mathrm{t}) = \boldsymbol{T}^t \cdot \boldsymbol{x}(0). \tag{15}$$

Equation (15) describes a Markov chain whose state transition matrix is $\mathbf{T}'$. The final rank of all pages in the graph is the stationary distribution $\boldsymbol{x}(\infty)$ of this chain. [Diligenti04] further shows that on some conditions there must exist such a distribution and that it is independent of the initial vector $\boldsymbol{x}(0)$.

IV.4.2.2 PageRank Calculation Based on a Random Walk

The single-surfer model may be extended to a *multisurfer walk* in which the things become slightly more complicated. In this model there are several surfers influencing one another. But we are interested more in how the PageRank calculation fits into the probabilistic single-surfer random walk framework.

PageRank is a special case of the single-surfer random walk in that it considers only two basic actions: jumping to a random page from the current page q with probability $x(j|q) = 1 - d$ and following a link from the page q with probability $x(l|q) = d$ (d may be chosen arbitrarily between 0 and 1 with no effect on the convergence). The other two probabilities known from the general model ($x(b|q)$ and $x(s|q)$) are null. Obviously, all the probabilities are independent of the current page q. The target page p of a jump is selected uniformly from all the N pages in the graph G, thus $x(p \mid j) = 1/N, \forall p \in G$. The probability of following a link from page q to page p is $x(p|q,l) = \alpha_q$ where $\alpha_q = 1/h_q$ and h_q is the *hubness* of page q, i.e. the number of links pointing from q elsewhere. Therefore we can rewrite (12) as

$$x_p(t+1) = \frac{1-d}{N}\sum_{q \in G} x_q(t) + d \sum_{q \in pa(p)} \alpha_q x_q(t) = \frac{1-d}{N} + d \sum_{q \in pa(p)} \alpha_q x_q(t) \tag{16}$$

where $\sum_{q \in G} x_q(t) = 1$.

The fact that $0 < d < 1$ implying $x(j|q) = 1 - d > 0$ guarantees that the PageRank vector converges to a distribution of scores independent of the initial distribution. Again, using a matrix notation, the computation of PageRank looks like this:

$$x_p(t+1) = \frac{1-d}{N} E + dW'\Theta x(t) \tag{17}$$

where $\boldsymbol{E}$ is the N × N identity matrix, $\boldsymbol{W}$ is the adjacency matrix of the Web graph (i.e. an element $(p,q) = 1$ if there is a link from p to q and it is zero otherwise), and $\boldsymbol{\Theta}$ is a diagonal matrix whose element $(q,q) = \alpha_q$.

There is a little problem with *sink* pages (compare with Figure IV.4) whose hubness is zero (i.e. $ch(q) = \varnothing$) and therefore we cannot compute the term $1/h_q$. Instead, it should be $x(l|q) = 0$ resulting in $x(j|q) = 1$ for any sink page q. So the PageRank equation must be modified in that $x(j|q) = 1 - d$ if $ch(q) \neq \varnothing$ and $x(j|q) = 1$ if $ch(q) = \varnothing$. Then the first term in (16) will not be constant but the probability $x(p \mid j,t) = \frac{1}{N}\sum_{q \in G} x(j \mid q) x_q(t)$ (jumping to p in time step t) needs to be computed at the beginning of each iteration.

IV.4.2.3 Conclusions

[Diligenti04] presents the HITS algorithm in terms of a multisurfer random walk notation (see Section III.1 for the original explanation) and compares HITS with PageRank: Computation of PageRank is stable and it can be applied to large document collections because small communities are not overwhelmed by large ones. On the other hand, PageRank does not take into account complex relationships of Web page citations. HITS is not stable, only the largest community influences the ranking but HITS understands better relations among pages. As a result, [Diligenti04] proposes a hybrid model called PageRank-HITS, which combines both of the algorithms.

Vertical ranking systems consider the contents of Web pages as well as the Web graph topology when assigning scores. Each page is represented by a set of keywords and it gets a relevance value by a classifier respecting the topic of interest. For instance, the page www.google.com is highly ranked by a general PageRank but it would be little ranked by a PageRank focused on the *data mining* topic. From a couple of focused ranking algorithms "double focused PageRank" turned out to be the best. We can easily imagine *applying a vertical ranking algorithm to finding experts in a given research domain*.

V Existing Systems for Support of Researchers

In the field of researchers support there exist a number of applications, mostly commercial, which combine classical data mining methods and visualization techniques. One exception is the Tétralogie project, a university based research work. Therefore, it is transparent and we will take a closer look at it further below.

V.1 Commercial Solutions

The following references to existing applications were adopted from the general work of Jean Archambeault [Archambeault02]. We present some interesting facts found on the corresponding Web sites.

One personal licence of VantagePoint [29] costs 7 500 USD. Documents must be found and imported to the program by hand. Entrieva [30] is able to collect the documents in various formats even on the Web. Among others, it creates taxonomies, locates changes and alerts the user. OmniViz [31] treats documents in various formats (e.g. ASCII text or Excel). Among others, it enables detecting co-workers and topics. The user may specify concepts which he/she considers more or less interesting. VxInsight [32] is a classical data mining tool of a governmental organization. Research publications are available. The Web site has been moved and it is in a strange state. MicroPatent [33] provides access to a database of primarily patent documents. For instance, it generates topographic maps of documents and terms and citation trees of patents.

ClearForest [34] is closest to what we would call a system for researchers support. Co-founder of the company is Dr. Ronen Feldman – claimed to be the father of text mining (see [Feldman95]). There are FBI, Kodak, Ford Motor etc. among its clients. An example of detection of relationships among terrorist organizations and terrorists is shown. Unstructured data are processed and concepts and relations are acquired from them. ClearForest offers several products. ClearResearch "helps researchers not waste their time reading published material. It provides names of key players, graphical and textual representation of their distribution, citations, time appearance, relations among members of an arbitrary category", etc. ClearEvents monitors the Web and it informs the user about business events in real time. Although white papers are available, almost no technical details are known, of course.

V.2 Free Services

Bibster [35] is an open source project for exchanging bibliographic information among researchers. It consists in a peer-to-peer network of scientists in which each participant has on his or her computer the same software. Once a researcher enters some bibliographic information into it, the information is made available to all other users of the system. This is especially useful when another researcher cites someone's publication and wants to find the exact annotation. Moreover, the list of publications maintained by a peer may reveal what is the domain of interest of that researcher. Of course, at the beginning some data must be typed in manually or it must be imported from existing BibTeX [36] files. Bibster makes use of ontologies and semantic Web technologies (see Section II.5 and I.6).

Another free tool often utilized by scholars is ResearchIndex [37] which we omitted in Section IV.2.2 on specialized search engines. Its services as well as the full source code are freely available. ResearchIndex uses search engines (with queries "publications", "papers", "postscript", etc.) and crawling to efficiently locate papers on the Web. Start points for

crawling may also be submitted by users who would like to have their documents indexed. It may take a few weeks after submitting to happen so. Its database is continuously updated 24 hours a day (it comprises over 700 000 documents primarily on computer science) and the citation index is constructed in a fully automated way – no manual effort is needed. Operating completely autonomously, ResearchIndex works by downloading papers in PDF or PS formats from the Web and converting them to text. It then parses the papers to extract the citations and the context in which the citations are made in the body of the paper, storing this information in a database. ResearchIndex includes full-text article and citation indexing, and allows the location of papers by keyword search or citation links. It can also locate papers related to a given article by using common citation information or word similarity. Given a particular paper, ResearchIndex can also display the context of how subsequent publications cite that paper [Lawrence99b].

When enumerating free services we should not forget Google Scholar [37] for searching research papers, Google Alerts [38] for being informed about breaking news on a certain topic and others.

V.2.1 Tétralogie

Tétralogie was initially a data mining tool for mining information from well-structured data such as bibliographic databases stored on a CD (e.g. [Hubert01]). It was meant to be used for science and technology monitoring (*scientific* or *technological watch*). The main operation is performed as a sequence of information filtering, reduction, mining and knowledge visualization processes. Tétralogie makes use of many techniques some of which we were talking about earlier: concept hierarchies (i.e. taxonomies, see Section II.3), candidate terms detection (i.e. repeated sequences, see Section II.2.1), *contingency tables*, *correspondence factorial analysis*, hierarchical agglomerative clustering (see Section II.3.1) and others.

A *contingency table* is a means of transforming non-numerical information into numerical information. Each *contingency table* is a 2D document representation. For example, a contingency table with rows representing the variable *country* and columns representing the variable *year* depicts a relative contribution of countries to publications in time provided we are analyzing a collection of research papers. Each cell in the table is the number of articles published in the corresponding country in a given year. *Contingency tables* serve as input to different mining modules after they have been reduced by using *factorial analysis* methods.

Tétralogie has a powerful visualization module that enables depicting collaborative networks, thematic maps, impact maps, topic evolution charts and others in as many as four dimensions. Tétralogie mines from local structured data. In the past, Tétralogie was successfully used to discover evolutionary trends in astronomical literature [Egret98], among others. The TétraFusion project [Crimmins99], which involved a cooperating Web metasearch engine and which allowed for Web mining, **has not been made publicly available**.

V.3 Conclusions

The topic of analyzing (Web) documents to support researchers is not new and several solutions may be found, particularly in the commercial area as common data mining applications. However, there is space for research in the public scientific field. Especially the functionality of ClearResearch, which is literally determined for researchers, and of Tétralogie might be a source of inspiration.

VI Goals of the Thesis

In the sections above we have presented some state-of-the-art knowledge and open issues in several domains of Web mining with emphasis on its applications to supporting researchers.

The aim of the doctoral thesis will be to create a Web mining support tool for researchers that would enable in particular:

- discovering new research topics, new researchers, new Web pages
- determining significance and activities of researchers
- visualizing time development of researchers and research groups as well as particular scientific areas

The principal means to achieve this will be mining (monitoring) from a flexible set of Web pages. We have shown that directions of some research effort may particularly be (without order of significance):

- experimenting with various patterns for term candidate generation (see Section II.2.1.2)
- finding best incremental hierarchical clustering methods for Web document taxonomies (see Section II.3.1.1)
- creating a detector of personal Web pages based on pronominal distributions (see Section II.4.1)
- experiment with different similarity measures between sets of elements (see Section II.5.1.2)
- combining Web authorities and "real" authorities to determine "universal" authorities (see Section III.2)
- applying a vertical ranking algorithm to finding experts in a given research domain (see Section IV.4.2.3)

The resulting prototype system will dynamically mine from unstructured (Web) data and it will be publicly accessible and open for the scientific community. (Let us recall that neither ClearResearch nor TétraFusion mentioned in Section V is free and publicly available.) One of the possible usage scenarios would be a user selecting a topic in a hierarchy of research areas, then choosing a scientist active in that field and receiving a list of publications as can be seen in FigureVI.1.

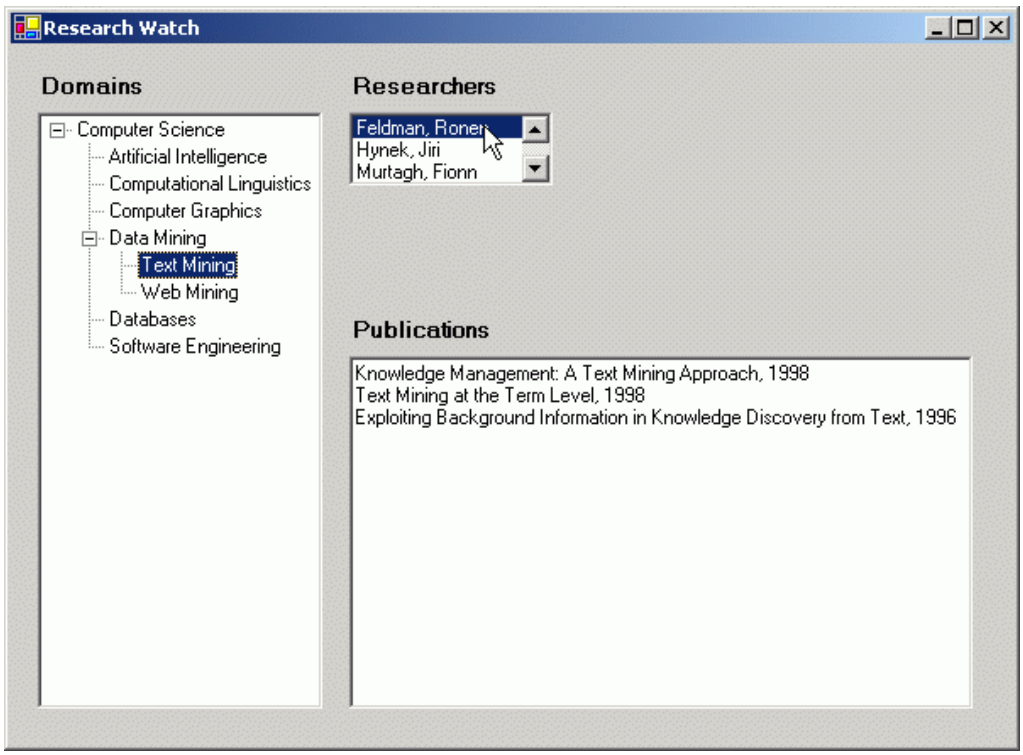

Figure VI.1: Sample system screenshot

References

Books and Papers

[Andersson95] Andersson A., Nilsson S.: "Efficient Implementation of Suffix Trees", *Software – Practice and Experience*, Vol. 25, No. 2, pp.129 – 141, 1995.

[Archambeault02] Archambeault J.: "Visualisation de l'évolution d'un domaine scientifique par l'analyse des résumés de publication à l'aide de réseaux neuronaux", MSc. thesis, Université de Montréal, 2002.

[Beaudouin01] Beaudouin V., Fleury S., Habert B., Illouz G., Licoppe C., Pasquier M.: "TyPWeb: décrir la Toile pour mieux comprendre les parcours", *Colloque International sur les Usages et les Services des Télécommunications*, pp. 492 – 503, 2001.

[Benjamins98] Benjamins R., Fensel D., Gomez-Perez A., Decker S., Erdmann M., Motta E., Musen M.: "Knowledge Annotation Initiative of the Knowledge Acquisition Community $(KA)^2$", *Proc. of Workshop on Knowledge Acquisition, Modeling and Management*, Banff, 1998.

[Brin98] Brin S., Page L.: "The Anatomy of a Large-Scale Hypertextual Web Search Engine", *Proceedings of the 7th World Wide Web Conference*, pp. 107 – 117, 1998.

[Chakrabarti02] Chakrabarti S.: "Mining the Web: Analysis of Hypertext and Semi Structured Data", Morgan Kaufmann, 2002.

[Chakrabarti98] Chakrabarti S., Dom B. E., Gibson D., Kumar R., Raghavan P., Rajagopalan S., Tomkins A.: "Spectral Filtering for Resource Discovery", *ACM SIGIR Workshop on Hypertext Information Retrieval on the Web*, 1998.

[Chisholm99] Chisholm E., Kolda T.G.: "New term weighting formulas for the vector space method in information retrieval", Technical report ORNL/TM-13756, Computer Science and Mathematics Division, Oak Ridge National Laboratory, 1999.

[Clifton04] Clifton C., Cooley R., Rennie J.: "TopCat: Data Mining for Topic Identification in a Text Corpus", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 8, pp. 949 - 964, 2004.

[Crimmins99] Crimmins F., Smeaton A. F., Dkaki T., Mothe J.: "TétraFusion: Information Discovery on the Internet", *IEEE Intelligent Systems*, Vol. 14., No. 4, pp. 55 – 62, 1999.

[Day97] Day D., Aberdeen J., Hirschman L., Kozierok R., Robinson P., Vilain M.: "Mixed-Initiative Development of Language Processing Systems", *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington, pp. 348 – 355, 1997.

[Diligenti04] Diligenti M., Gori M., Maggini M.: "A Unified Probabilistic Framework for Web Page Scoring Systems" , *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 1, pp. 4 – 16, 2004.

[Egret98] Egret D., Mothe J., Dkaki T., Dousset B.: "Information mining in astronomical literature with Tétralogie", *Proc. of the 7th annual conference on Astronomical Data Analysis Software and Systems*, pp. 461 – 465, 1998.

[Feldman95] Feldman R., Dagan I.: "Knowledge Discovery in Textual Databases (KDT)", *Proceedings of the First International Conference on Knowledge Discovery*, Montreal, pp. 112 – 117, 1995.

[Feldman96] Feldman R., Dagan I., Kloesgen W.: "Efficient Algorithm for Mining

and Manipulation Associations in Texts", *Proceedings of the 13th European Meeting on Cybernetics and Research*, Vienna, pp. 949 – 954, 1996.

[Feldman97a] Feldman R., Hirsh H.: "Exploiting Background Information in Knowledge Discovery from Text", *Journal of Intelligent Information Systems*", Vol. 9, No. 1, pp. 83 – 97, 1997.

[Feldman97b] Feldman R., Aumann Y., Amir A., Kloesgen W., Zilberstien A.: "Maximal Association Rules: a New Tool for Mining for Keyword co-occurrences in Document Collections", *Proc. of the 3rd Int. Conf. on Knowledge Discovery*, Newport Beach, CA, pp. 167 - 170, 1997.

[Feldman98a] Feldman R., Fresko M., Hirsh H., Aumann Y., Liphstat O., Schler Y., Rajman M.: "Knowledge Management: A Text Mining Approach", *Proceedings of the 2nd International Conference on Practical Aspects of Knowledge Management*, 1998.

[Feldman98b] Feldman R., Fresko M., Kinar Y., Lindell Y., Liphstat O., Rajman M., Schler Y., Zamir O.: "Text Mining at the Term Level", *Proceedings of Principles of Data Mining and Knowledge Discovery*, pp. 65 – 73, 1998.

[Flynn04] Flynn C., Dunnion J.: "Event Clustering in the News Domain", *Proceedings of TSD 2004*, pp. 65 – 72, 2004.

[Gibson98] Gibson D., Kleinberg J., Raghavan P.: "Inferring Web Communities from Link Topology", *Proceedings of the 9th ACM Conference on Hypertext and Hypermedia*, pp. 225 – 234, 1998.

[Grabowski99] Grabowski S.: "Text preprocessing for Burrows-Wheeler block sorting compression", *Proc. Sieci i Systemy Informatycne - teoria, projekty, wdrozenia*, Lodz, 1999.

[Grolmus03a] Grolmus P., Hynek J., Jezek K.: "User Profile Identification Based on Text Mining", *Proc. of 6th Int. Conf. ISIM'03*, MARQ Ostrava, pp. 109 - 118, 2003.

[Grolmus03b] Grolmus P., Hynek J., Jezek K.: "Vyhledavani castych frazi pro generovani uzivatelskych profilu", *ITAT*, 2003.

[Hammouda04] Hammouda K., Kamel M.: "Efficient Phrase-Based Document Indexing for Web Document Clustering", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 10, pp. 1279 – 1296, 2004.

[Han00] Han J., Kamber M.: "Data Mining: Concepts and Techniques", Morgan Kaufmann Publishers, 2000.

[Hand01] Hand D., Mannila H., Smyth P.: "Principles of Data Mining" , MIT Press, Cambridge, MA, 2001.

[Hubert01] Hubert G., Mothe J., Benammar A., Dkaki T., Dousset B., Karouach S.: "Textual Document Mining Using a Graphical Interface", *Human computer interaction International*, Vol. 1, pp. 918 – 922, 2001.

[Justeson95] Justeson J., Katz M.: "Technical terminology: some linguistic properties and an algorithm for identification in text", *Natural Language Engineering*, Vol. 1, No. 1, pp. 9 - 27, 1995.

[Kosala00] Kosala R., Blockeel H.:, "Web Mining Research: A Survey", *ACM SIGKDD Explorations Newsletter*, Vol. 2, No. 1, pp. 1 – 15, 2000.

[Lawrence99a] Lawrence S., Giles C. L.: "Accessibility of Information on the Web", *Nature*, Vol. 400, pp. 107 – 109, 1999.

[Lawrence99b] Lawrence S., Giles C. L., Bollacker K.: "Digital Libraries and Autonomous Citation Indexing", *IEEE Computer*, Vol. 32, No. 6, pp. 67 – 71, 1999.

[Lebart94] Lebart L., Salem A.: "Statistique textuelle", Dunod, 1994.

[Mitra97] Mitra M., Buckley C., Singhal A., Cardie C.: "An Analysis of Statistical and Syntactic Phrases", *Proceedings of the 5th International Conference on Computer-Assisted Information Searching on the Internet*, pp. 200 - 214, Montreal, 1997.

[Nassar04] Nassar S., Sander J., Cheng C.: "Incremental and Effective Data Summarization for Dynamic Hierarchical Clustering", *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, Paris, pp. 467 – 478, 2004.

[Nevill-Manning97] Nevill-Manning C. G., Witten I. H.: "Compression and Explanation Using Hierarchical Grammars", *The Computer Journal*, Vol. 40, No. 2/3, pp. 103 – 116, 1997.

[Oueslati96] Oueslati R., Frath P., Rousselot F.: "Term Identification and Knowledge Extraction", *Proc. of Int. Conference on Applied Natural Language and Artificial Intelligence*, Moncton, 1996.

[Page98] Page L., Brin S., Motwani R., Winograd T.: "The PageRank Citation Ranking: Bringing Order to the Web", Technical Report, Comp. Science Dept., Stanford University, 1998.

[Piatetsky-Shapiro91] Piatetsky-Shapiro G., Frawley W. (Eds.): "Knowledge Discovery in Databases", AAAI Press, Menlo Park, CA, 1991.

[Rajman98] Rajman M., Besançon R.: "Text Mining – Knowledge Extraction from Unstructured Textual Data", *Proceedings of the 6th Conference of International Federation of Classification Societies*, 1998.

[Richy02] Richy H.: "Métadonnées et documents numériques", *Technique de l'Ingénieurs, traité Informatique*, pp. 1 – 14, 2002.

[Sandeep04] Sandeep T., Hankins R. A., Patel J. M.: "Practical Suffix Tree Construction", *Proc. of Conf. on VLDB 2004*, pp. 36 - 47, 2004.

[Shakes97] Shakes J., Langheinrich M., Etzioni O.: "Dynamic Reference Sifting: A Case Study in the Homepage domain", *Proc. of the 6th WWW Conf.*, 189 – 200, Santa Clara, CA, 1997.

[Srikant95] Srikant R., Agrawal R.: "Mining generalized association rules", *Proc. of the 21st VLDB Conference*, Zurich, pp. 407 – 419, 1995.

[Stubbs04] Stubbs M.: "On very frequent phrases in English: distributions, functions and structures", *25th anniversary meeting of ICAME*, Verona, 2004.

[Ukkonen95] Ukkonen E.: "On-line construction of suffix-trees", *Algorithmica*, Vol. 14, No. 3, pp. 249 – 260, 1995.

[Varlamis04] Varlamis I., Vazirgiannis M., Halkidi M., Nguyen B.: "THESUS, a Closer View on Web Content Management Enhanced with Link Semantics", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 6, pp. 685 – 700, 2004.

[Weiner73] Weiner P.: "Linear pattern matching algorithm", *Proceedings of the 14th IEEE Symposium on Switching and Automata Theory*, New York, pp. 1 - 11, 1973.

[Zamir98] Zamir O., Etzioni O.: "Web document clustering: A feasibility demonstration", *Proc. of the 21st Annual Int. ACM SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, pp. 46 – 54, 1998.

[Zipf49] Zipf G. K.: "Human Behaviour and the Principle of Least-Effort", Addison-Wesley, Cambridge MA, 1949.

Internet

[1] http://www2.cs.uregina.ca/~hamilton/courses/831/notes/kdd/kdd.gif
[2] RDF Site Summary (RSS) 1.0: http://web.resource.org/rss/1.0/
[3] TDT: http://www.nist.gov/speech/tests/tdt/
[4] L'outil de traitement de corpus LIKES: http://www-ensais.u-strasbg.fr/liia/LIIA_Products_Installers/install.htm
[5] Corvigo: http://www.corvigo.com/media/news/20040102.html
[6] Google Directory: http://www.google.com/dirhp
[7] Yahoo!: http://www.yahoo.com/
[8] Wikipedia: http://www.wikipedia.org/
[9] research-topics ontology: http://hcs.science.uva.nl/usr/richard/ka2/research-topic.html
[10] Netcraft: Web Server Survey Archives: http://news.netcraft.com/archives/web_server_survey.html
[11] Kansas City Public Library - Introduction to Search Engines: http://www.kclibrary.org/resources/search/intro.cfm
[12] Google Review on Search Engine Showdown: http://www.searchengineshowdown.com/features/google/review.html
[13] Search Engine Showdown Reviews: http://searchengineshowdown.com/reviews/
[14] Google: http://www.google.com/
[15] Robot Exclusion Standard: http://www.robotstxt.org/wc/exclusion.html
[16] Database of Web Robots, Overview: http://www.robotstxt.org/wc/active/html/index.html
[17] Freshness Showdown: http://www.searchengineshowdown.com/stats/freshness.shtml
[18] Search Engines Showdown: Size Comparison Methodology: http://www.searchengineshowdown.com/stats/methodology.shtml
[19] Online JOurnal Search Engine: http://ojose.lanners.org/
[20] Scirus – for scientific information: http://www.scirus.com/
[21] Welcome to Phibot: http://phibot.org/new/Search/Main
[22] ISI Web of Science: http://www.isinet.com/isi/products/citation/wos/index.html
[23] Testa J.: The ISI Database: The Journal Selection Process, http://sunweb.isinet.com/isi/hot/essays/selectionofmaterialforcoverage/199701.html
[24] Testa J.: Current Web Contents: Developing Web Site Selection Criteria, http://sunweb.isinet.com/isi/hot/essays/selectionofmaterialforcoverage/23.html
[25] Science Citation Index help, Web version: Princeton University: http://www.princeton.edu/~biolib/instruct/SCI.html
[26] Computer Science Papers NEC Research Institute CiteSeer Publications ResearchIndex: http://citeseer.nj.nec.com/cs
[27] RFC 1950: http://www.faqs.org/rfcs/rfc1950.html
[28] Google Toolbar: http://toolbar.google.com/googlebar.html
[29] VantagePoint: http://www.thevantagepoint.com/index.html
[30] Entrieva: http://www.semio.com/
[31] OmniViz: http://www.omniviz.com/
[32] VxInsight: http://www.cs.sandia.gov/projects/VxInsight.html
[33] MicroPatent: http://www.aurigin.com/
[34] ClearForest: http://www.clearforest.com/
[35] Bibster: http://bibster.semanticweb.org/
[36] The BibTeX Format: http://www.ecst.csuchico.edu/~jacobsd/bib/formats/bibtex.html
[37] Google Scholar: http://scholar.google.com/
[38] Google Alerts: http://www.google.com/alerts
[39] Gedcom: http://www.genealogie-standard.org/logiciel/gedcom.html
[40] Open Directory Project: http://dmoz.org/about.html

Author's Publications

[Tesar05] Tesar R., Fiala D., Rousselot F., Jezek K.: "A comparison of two algorithms for discovering repeated word sequences" (accepted for publication at *Data Mining 2005*)

Belaïd A., Alusse A., Rangoni Y., Cecotti H., Farah F., Gagean N., Fiala D., Rousselot F., Vigne H.: "Document retro-conversion for personalized electronic reedition", *IWDA'05*, Calcutta, 2005.

Fiala D., Jezek K.: "Retrieving citations on the Web", *Proceedings of International Conference on Knowledge Engineering and Decision Support ICKEDS'04*, Porto, pp. 481 – 488, 2004.

[Fiala03] Fiala D.: "A System for Citations Retrieval on the Web", MSc. thesis, University of West Bohemia in Pilsen, 2003.

List of Acronyms

ANTF	Augmented Normalized Term Frequency
ASCII	American Standard Code for Information Interchange
CD	Compact Disk
CPU	Central Processing Unit
DAML	DARPA Agent Markup Language
DARPA	Defense Advanced Research Projects Agency
DRS	Dynamic Reference Sifting
FBI	Federal Bureau for Investigation
FP	Frequent Pattern
HAC	Hierarchical Agglomerative Clustering
HITS	Hyperlink-Induced Topic Search
HP	Hewlett-Packard
HTML	HyperText Markup Language
IDF	Inverse Document Frequency
IP	Internet Protocol
IR	Information Retrieval
ISI	Institute for Scientific Information
KA2	Knowledge Annotation Initiative of the Knowledge Acquisition Community
KDD	Knowledge Discovery in Databases
LIKES	LInguistic and Knowledge Engineering Station
MSN	Microsoft Network
OIL	Ontology Inference Layer
OJOSE	Online JOurnals Search Engine
PDF	Portable Document Format
PS	PostScript
RAM	Random Access Memory
RDF	Resource Description Framework
RS	Repeated Segments
RSS	RDF Site Summary
RTF	Rich Text Format
SHC	Similarity Histogram-based Clustering
SHOE	Simple HTML Ontology Extensions
SHTML	Server-side HTML
ST	Suffix Tree
TDT	Topic Detection and Tracking
THESUS	THEmatic SUbsetS
URL	Uniform Resource Locator
USB	Universal Serial Bus
XML	eXtended Markup Language