AN ALGORITHM FOR CLUSTERING OF WEB SEARCH RESULTS

by
STANISŁAW OSIŃSKI

Supervisor
JERZY STEFANOWSKI, Assistant Professor

Referee
MACIEJ ZAKRZEWICZ, Assistant Professor

MASTER THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science,
Poznań University of Technology, Poland

June 2003

ALGORYTM GRUPOWANIA WYNIKÓW ZAPYTAŃ DO WYSZUKIWAERK INTERNETWYCH

STANISŁAW OSIŃSKI

Supervisor
dr hab. inż. JERZY STEFANOWSKI

Referee
dr inż. MACIEJ ZAKRZEWICZ

PRACA MAGISTERSKA

Projektowanie i Eksploatacja Systemów Informatycznych
Instytut Informatyki, Politechnika Poznańska,

czerwiec 2003

Abstract

In this thesis we propose a description-oriented algorithm for clustering of results obtained from Web search engines called LINGO. The key idea of our method is to first discover meaningful cluster labels and then, based on the labels, determine the actual content of the groups. We show how the cluster label discovery can be accomplished with the use of the Latent Semantic Indexing technique. We also discuss several factors that influence the quality of cluster description, such as input data preprocessing and language identification.

As part of this thesis we prepared an implementation of our algorithm within the framework of Carrot^2. To evaluate the practical performance of our algorithm we propose an expert-based scheme for assessment of clustering results.

Streszczenie

W niniejszej pracy magisterskiej proponujemy algorytm grupowania wyników zapytań skierowanych do internetowych serwisów wyszukiwawczych, którego najważniejszym celem jest zapewnienie wysokiej jakości opisu utworzonych skupień. Podstawową ideą proponowanego algorytmu jest rozpoczęcie procesu grupowania od ustalenia zrozumiałych dla użytkownika opisów tworzonych grup. W kolejnej fazie algorytmu znalezione uprzednio opisy stają się podstawą do przydziału wyników zapytania do odpowiednich skupień. W pracy tej pokazujemy w jaki sposób technika Latent Semantic Indexing może być wykorzystania podczas poszukiwania odpowiednich opisów grup. Wskazujemy również inne elementy wpływające na jakość opisu sukupień takie jak wstępne przetwarzanie danych oraz rozpoznawanie języka w którym zostały napisane grupowane dokumenty.

Częścią prezentowanej pracy jest implementacja proponowanego algorytmu w środowisku Carrot^2. Dla potrzeb oceny praktycznej wydajności stworzonego algorytmu proponujemy ekspercką metodę oceny jakości wyników grupowania.

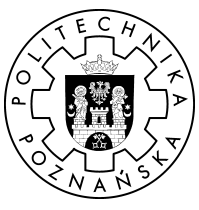
Acknowledgments

During the course of writing my thesis I had the genuine pleasure to work with a number of people without whose support completion of this project would be barely possible. First of all, I wish to express my sincere appreciation to Mr. Jerzy Stefanowski for enlivening my interest in web search results clustering and for the guidelines and advice he gave me. I would also like to thank Mr. Maciej Zakrzewicz, the reader of my work, for his valuable insights and observations.

I wish to pay special gratitude to Mr. Dawid Weiss, Paweł Kowalik and Michał Wróblewski – the developers of the Carrot^2 framework. Not only did Carrot^2 help us put our ideas into action, but it first of all gave us the opportunity to create a close-knit and highly successful software team. Let us keep up the good work, friends!

I would also like to thank Agata Godlewska, Paweł Kowalik, Krystian Nowak, Jerzy Stefanowski, Dawid Weiss and Michał Wroblewski for their time and effort devoted to evaluation of LINGO.

Finally, it is difficult to explain how grateful I am to my nearest and dearest whose endless patience and personal involvement helped me to complete this challenging venture. Never before did I need you so much.



Typographical errors, mistakes and any other comments are welcome and should be directed to Stanisław Osiński, stachoo@man.poznan.pl. In case this document is used as a reference, we kindly ask to notify the author at the e-mail address above.

 This document has been written using DocBook DTD standard and formatted with RenderX XEP Academic Edition. Use high-resolution laser printer to obtain sharp figures.

Table of Contents

List of Figures

List of Tables

1 Introduction

Knowledge is of two kinds: we know a subject ourselves, or we know where we can find information about it.

--Samuel Johnson, 1775

For centuries, the words of Samuel Johnson remained true to those in search for information. Although not always readily available, information could be easily found by individuals in libraries and archives, carefully organised and indexed by human experts. The knowledge of where to find information was perfectly enough to be successful. With the advent of the Internet, however, the situation changed dramatically. Enormous amounts of data are now freely accessible to over 600 millions of users on-line. Unfortunately, only a mere fraction of this group is able find appropriate shelves and books in this huge library-without-indices-and-librarians. The knowledge of *where* to seek for information turned out to be not enough. Should Johnson live in the 21st century, he would undoubtedly say: "[…] we know a subject ourselves, or we know *how* we can find information about it".

1.1 Motivation

The question of how to find information of interest in the Internet is raised by the **Web Search Problem** [Selberg, 99]: *find the set of documents on the Web relevant to a given user query*. The definition differs from the well-known **Information Retrieval Problem**: *given a set of documents and a query, determine the subset of documents relevant to the query*, in several aspects. First of all, it recognises the fact that the Web is a highly dynamic collection of documents[1], which makes many of the classic indexing schemes unsuitable. Secondly, due to the sheer size of the Internet, the Web Search Problem assumes that only a limited number of documents will actually be matched against the input query and even a smaller proportion of them will be finally viewed by the user. Thus, special attention must be given to helping the user choose the most relevant documents first.

The query-list paradigm

To solve the problem of relevance judgement most of the currently available search engines, such as Google [Google, 03], AllTheWeb [AllTheWeb, 03] or Yahoo [Yahoo, 03], adopt a so-called **query-list paradigm**. In response to the user query (usually a list of words) the

[1] in 2001 Google informed that they had indexed over 1.35 billion pages; in 2003 the number soared to over 3.08 billion

system returns a ranked list of documents that match the query – the higher on the list, the higher the relevance. Many schemes for sorting of the ranked list have been proposed, ranging from variations of simple similarity calculation [Yuwono and Lee, 96] to complex graph analysis [Google, 03][Kleinberg, 98]. While such schemes work well with queries that are precise and narrow, if the query is too general, it is extremely difficult for the search engine to identify the specific documents in which the user was interested. As a result, to find the requested information the user is made to sift through a long list of irrelevant documents. Such a type of search is called a **low precision search** [Zamir, 99].

Tackling low precision searches

As noted in [Bharat and Henzinger, 98] and [Weiss, 01], the majority of queries directed to the Internet search engines are general, 1-3 words in length. Low precision searches are therefore inevitable and methods of dealing with the results of such searches are needed. One method is filtering of the ranked list of documents, varying from simple pruning techniques to advanced Artificial Intelligence algorithms. Although they limit the total length of the ranked list, filtering methods not always help the users locate the specific documents they searched for. Another method is **relevance feedback** where the search engine helps the user refine their query by adding more specific keywords to it. One more method, implemented e.g. in Yahoo, is including in the ranked list a number of documents from a human-made directory. The drawback of this method is that due to the high dynamics of the World Wide Web, many of the documents from the catalogue may be outdated or may not be available anymore. Alternatively, experimental methods have been proposed that use map-like [MapNet, 03] or graph-like [Kartoo, 03] visualisation of the search results. Finally, clustering seems to be a promising approach to making the search results more understandable to the users.

Web Search Results Clustering

Clustering is a process of forming groups (clusters) of similar objects from a given set of inputs. When applied to web search results, clustering can be perceived as a way of organising the results into a number of easily browsable thematic groups. For a query "clinton", for example, the results could be presented in such topical groups as "Bill Clinton", "Hillary Clinton", "George Clinton" and "Clinton County". In this way, the inexperienced users, who may have difficulties in formulating a precise query, can be helped in identifying the actual information of interest. Moreover, the advanced users, who sometimes issue general queries to learn about the whole spectrum of sub-topics available, will no longer be made to manually scan hundreds of the resulting documents. Instead, they will be presented with a concise summary of various subjects, with an option of drilling down selected topics.

What should be emphasised about web search clustering is that the thematic groups must be created *ad hoc* and fully automatically. Additionally, in contrast to the classic Information Retrieval tasks, the web search clustering algorithms do not operate on the full text documents, but only on the short summaries of them returned by search engines, called **snippets**. Therefore, the algorithms must take into account the limited length and often low quality of input data [Zamir, 99].

Several clustering engines for web search results have been implemented. Grouper [Zamir and Etzioni, 99] employs a novel, phrase-based algorithm called Suffix Tree Clustering

(STC), in Carrot [Weiss, 01] this kind of algorithm is applied to clustering of documents in Polish. Other examples of clustering engines can be the Scatter/Gather [Hearst and Pedersen, 96] system, the Class Hierarchy Construction Algorithm [Shenker et al., 01] and iBoogie [iBoogie, 03]. Vivisimo [Vivisimo, 02a] uses an intriguing, yet not publicised, technique for organising the search results into hierarchical and very well described thematic groups.

Vivisimo is an excellent example of how important for the overall quality of clustering are readable and unambiguous descriptions (labels) of the thematic groups. They provide the users with an overview of the topics covered in the results and help them identify the specific group of documents they were looking for. Unfortunately, the problem of the quality of group descriptions seems to have been neglected in previously proposed algorithms. Very often, overlong and ambiguous group labels are produced, which makes it difficult for the users to decide whether the group is noteworthy or not. Also, when clustering a set of documents in a mixture of different languages, sometimes completely meaningless group labels happen to be produced. The goal of this work is therefore to propose a new algorithm in which the high quality of clustering would be attained by ensuring that all discovered thematic groups are properly described.

1.2 The goal and scope of work

The goal of work

The main goal of this thesis is to propose a description-oriented algorithm for clustering of web search results. The following sub-tasks comprise the main goal:

a. **A study of recent Information Retrieval and Web Mining literature**

   The study should examine current trends in Information Retrieval, with special emphasis placed on Web Content Mining and web search results clustering. Applications of linear algebra to Information Retrieval, including Vector Space Model and Latent Semantic Indexing, should also be reviewed.

b. **Design of a description-oriented algorithm for clustering of web results**

   An attempt should be made to propose an automatic on-line web search results clustering algorithm. The algorithm should aim at producing concise, readable and meaningful descriptions of the discovered groups.

c. **Implementation of the proposed algorithm within the the framework of Carrot^2**

   The proposed algorithm should be implemented within the framework of Carrot^2 [Weiss, 03]. The implementation should enable experiments with different settings of the algorithm's parameters.

d. **Empirical evaluation of the proposed algorithm**

   The evaluation of the algorithm should comprise a study of clustering results for a number of sample real-world queries as well as a limited end-user feedback presentation.

Place of this thesis in the domain of Computer Science

This thesis addresses problems belonging to the area of **Web Mining** – a sub-field of Information Retrieval that deals with analysing data available in the Internet. Web Mining itself can be divided into a number of sub-areas: Web Usage Mining which involves automatic discovery of patterns in user access to web servers, Web Structure Mining which concentrates on the analysis of the hyperlink structure of the Web, and finally, Web Content Mining which studies problems of retrieving information from the Internet. It is the latter sub-area to which this thesis is closest.

1.3 Thesis structure

The remaining part of this thesis is organised as follows. We begin Chapter 2 with a brief introduction to Internet search services. We examine different approaches to the presentation of web search results, with special attention given to clustering interfaces. We continue the chapter with an outline of different text clustering algorithms, discussing the text processing techniques involved.

Chapter 3 introduces the concepts used in our proposed web search results clustering algorithm. We start with explaining two linear algebra models for Information Retrieval: Space Vector Model and Latent Semantic Indexing. Then, we proceed with a description of suffix arrays – a data structure facilitating a wide variety of text processing operations.

Chapter 4 explains the details of LINGO – our proposed description-oriented clustering algorithm. We present all stages of the algorithm, giving simplified examples where necessary. Then, we briefly address the implementation and efficiency issues. We conclude the chapter with an analysis of the strong and weak points of LINGO, identifying the areas of future improvements.

In Chapter 5 we present the results obtained from LINGO in response to a number of real-world queries, comparing them to those produced by other clustering algorithms. We also report on user feedback regarding LINGO's performance.

Chapter 6 concludes this thesis and gives directions for future work.

1.4 Typographic conventions

A number of typographic conventions are used throughout this thesis to make the reading of the text easier. Words or phrases *in italics* are particularly important for proper understanding of the surrounding context and should be paid special attention to. Italics are also used to place emphasis on a word. First occurrences of terms or definitions will be denoted by **bold face**.

Margin notes and references

Margin notes are used to mark important concepts being discussed in the paragraphs next to them. They are also to help the Reader scan the text. Reference numbers of all figures in

the text are composed of the number of the chapter they appear in and the consecutive number of the figure within the chapter. Thus, the fifth figure in the fourth chapter would be referenced as Figure 4.5. Square brackets denote citations and references to other articles, books and Internet resources listed at the end of this thesis.

2 Searching the Web

In this chapter we introduce problems of searching the Internet along with the associated text processing techniques. We start with explaining the purpose and the internal structure of web search engines. Then we proceed with a survey of the most common approaches to the presentation of web search results, with special emphasis placed on clustering interfaces. We finish this chapter with a review of a number of text clustering algorithms and the text processing techniques involved.

2.1 Web search services

The primary goal of **web search services**, also referred to as **web search engines**, is to help the Internet users locate resources of interest on the Web. Most commonly, the resources are of textual type and include HTML[2] pages, PDF[3], PS[4], MS Word and MS PowerPoint documents.

The overwhelming majority of web search engines are accessible through the HTTP[5] protocol and have their own HTML pages. Thus, to make a web search, the user points their web browser to one of the search services (e.g. to *http://www.google.com*, see Figure 2.1) and enters a **query**, which is usually a list of keywords that describe the information for which the user is looking. As a result, a list is returned which contains hyperlinks to the documents that are most likely to contain the requested information. Very often the hyperlinks are accompanied by short excerpts from the source documents called **snippets**.

Specialised search services

Due to the endless variety of the Internet's contents, specialised search engines proliferate, which help the users locate resources of a very specific type. Among the most popular of such services are: Google Groups [Google Groups, 03] – a Usenet search engine, AllTheWeb FTP [AllTheWeb FTP, 03], which finds files stored on FTP[6] servers, CiteSeer [CiteSeer, 03], which aids scientific paper searches. Also, very narrowly specialised services are available, Froogle [Froogle, 03] - a web shopping search engine, or Questlink [Questlink, 03] - an electronic parts search engine being only two of many examples.

---

[2] HyperText Markup Language

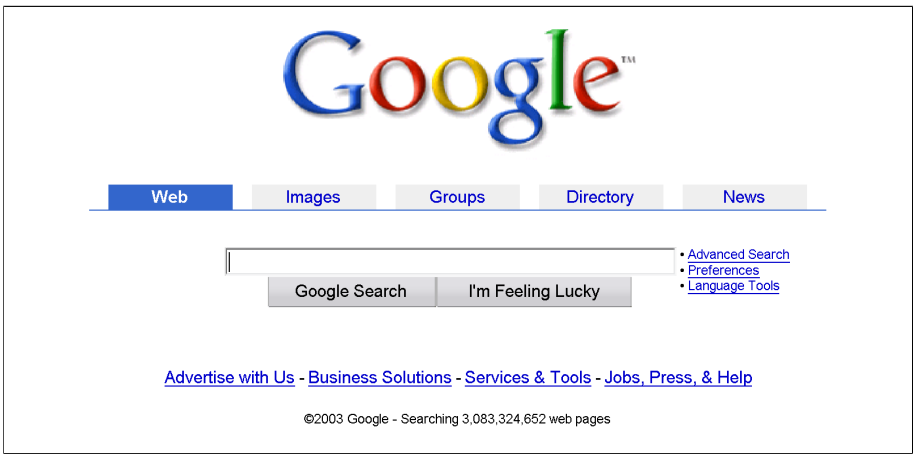
[3] Portable Document File

[4] Post Script

[5] HyperText Transfer Protocol

[6] File Transfer Protocol

Figure 2.1
Google home page



Meta-search engines

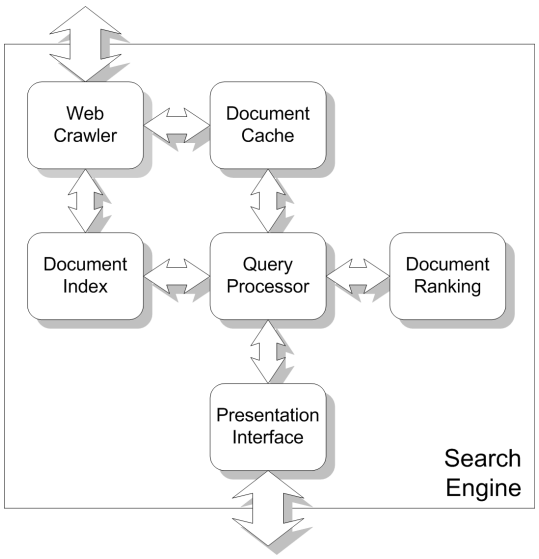
Another type of search engine is a so called **meta-search engine**, which directs the user's query to other search services, merges thus obtained results and sends them back to the user. The rationale behind such approach is that if no single search service is able to index the whole Web, then maybe it is sensible to combine their coverage, which might extend the searchable area [Selberg, 99]. Among the most popular meta-search engines are: Metacrawler [Metacrawler, 03], Dogpile [Dogpile, 03] and Mamma [Mamma, 03].

In contrast to the great diversity in the indexed content, the anatomy of modern search services remains similar and comprises a set of common components such as a web crawler, document index or query processor. The following section describes them in detail.

2.1.1 Internal structure

In Figure 2.2 components of a typical search engine are shown, which include: web crawler, document cache, document index, document ranking, query processor and presentation interface [Sherman, 02]. The components represent the logical structure of a search service rather than the implementation one, thus different physical architectures are possible (e.g. distributed or centralised).

Figure 2.2
Components of a typical search engine



Web Crawler

One of the two components directly interacting with the Internet is the web crawler, which is often called a web spider or robot. Its major role is to automatically discover new resources on the Web in order to make them searchable. Such process is defined by the **Web Discovery Problem** [Selberg, 99]: *How can all Web pages be located?* Web crawlers solve the problem by recursively following hyperlinks obtained from the already visited pages:

Figure 2.3 Web crawler algorithm pseudo-code [Selberg, 99]

```
/**
 * An example web crawler algorithm
 *
 * urlPool       is a set of internet addresses initially containing
 *               at least one URL
 * documentIndex is a data structure that stores information about
 *               the contents of the crawled pages
 */
webCrawler(UrlPool urlPool, DocumentIndex documentIndex)
{
  while (urlPool not empty)
  {
    url = pick URL from urlPool;
    doc = download url;
    newUrls = extract URLs from doc;
    insert doc into documentIndex;
    insert url into indexedUrls;

    for each u in newUrls
    {
      if (u not in indexedUrls)
      {
        add u to urlPool
      }
    }
  }
}
```

Apart from discovering new resources in the Internet, web crawlers check whether the already indexed pages are still available and whether they have been updated. This must be done in order for the search engine to keep an up-to-date snapshot of the Web contents, which is known as the **Web Coherence Problem** [Selberg, 99]. In practical implementations, the frequency of revisiting the already indexed pages depends on how often they are updated – news sites, for instance, may need indexing on daily or even hourly basis.

Robot Exclusion Protocol

As many pages on the Web are nowadays generated dynamically, for many reasons it may be undesirable that web spiders index their contents. Therefore, site administrators can use Robot Exclusion Protocol [REP, 03] to prevent certain pages from being crawled.

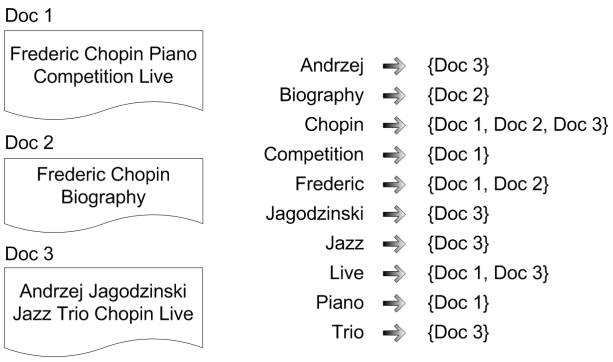
Document Index

The primary aim of the document index is to aid searches of the type: *Which documents contain the words given?* To answer such a question the majority of search engines use variants of a data structure called **inverted index**. Inverted index is much like the index listings in the back of a book – for every word a list of pointers is kept to the pages where the word occurs (see Figure 2.4).

A variant of an inverted index is also possible where pointers to the actual occurrences of a word within a document are stored. With such an index, the search engine can perform

phrase searches or **proximity searches** by finding words that are near each other in a document.

Figure 2.4 *An example inverted index*



In the World Wide Web environment the total size of the documents to be indexed can reach hundreds of terabytes. This poses many problems, especially in terms of storing and searching the index. State of the art indexing techniques can reduce the size of an inverted index to about 30% of the size of the text, with further reductions possible (up to 10% of the original text) when data compression is used [Baeza-Yates et al., 99]. Efficiency considerations regarding creation and searching of inverted indices are presented in [Baeza-Yates et al., 99].

Document Cache

Many search services apart from maintaining the document index store the indexed resources in their original form. In Google, the full text of the indexed pages is used to extract snippets and to serve the cached versions of documents [Sherman, 02]. The latter turns out to be an extremely advantageous feature, especially when the source document is temporarily or permanently unavailable.

Document Ranking

Inevitably, in the World Wide Web environment, even for fairly precise queries, thousands or millions of matching documents are reported. Techniques are therefore needed that will rank the retrieved documents so that the most relevant and most interesting of them are returned first.

Vector model based ranking

Historically, to rank the matching documents, search engines employed measures based on Vector Space Model. According to the model, both documents and queries are represented by multidimensional vectors, each dimension corresponding to one of all the distinct words present in the indexed documents. The relevance of a document to the query is then calculated as a dot product between the appropriate document vector and the query vector (for more details about the Vector Space Model refer to Chapter 3). Although this ranking scheme works well in classic Information Retrieval tasks, in the World Wide Web environment such relevance measure can be easily manipulated by including in a document all possible keywords. Fortunately, modern search engines use hyperlink-based ranking algorithms, which are far less susceptible to manipulation.

Hyperlink-based ranking

Hyperlink-based ranking algorithms utilise information about the relationships between web documents. The underlying assumption is that the number of hyperlinks that point to a page provides a measure of its popularity and quality. The following description of PageRank [Page et al., 98] – the hyperlink-based ranking algorithm used by Google, is given in [Baeza-Yates et al., 99]:

Google's document ranking algorithm [Baeza-Yates et al., 99]

> *The algorithm simulates a user navigating randomly in the Web who jumps to a random page with probability q or follows a random hyperlink (on the current page) with probability 1-q. It is further assumed that the user never goes back to the previously visited page following an already traversed hyperlink backwards. This process can be modelled with a Markov chain, from where the stationary probability of being in each page can be computed. This value is then used as part of the ranking mechanism.* [Baeza-Yates et al., 99]

An important yet non-technical issue connected with document ranking is the influence of commercial or government organisations. Google claims that it has *a "wall" between the search relevance ranking and advertising: no one can pay to be the top listing in the results, but the sponsored link spots are available* [Sherman, 02].

Query Processor

The role of the query processor is to coordinate the execution of the user's query. During the execution, query processor communicates with document index, cache and ranking components in order to obtain a list of documents that are relevant to the query. The list is then passed to the presentation interface, which will send it back to the user.

Presentation Interface

Search results presentation interfaces play an important role in the process of searching the web and have great influence on the overall quality of a search engine. For this reason we have decided to devote a separate subsection to them.

2.1.2 Search results presentation

The role of the search results presentation interface is to display the results in a way that helps the users identify the specific documents they searched for. Thus, an ideal interface should not make the users stumble upon the documents that *they* would judge as irrelevant. The task is particularly difficult due to the fact that most users formulate short and very general queries [Bharat and Henzinger, 98] [Weiss, 01], giving the search engine no clues about the specific subject they are interested in.

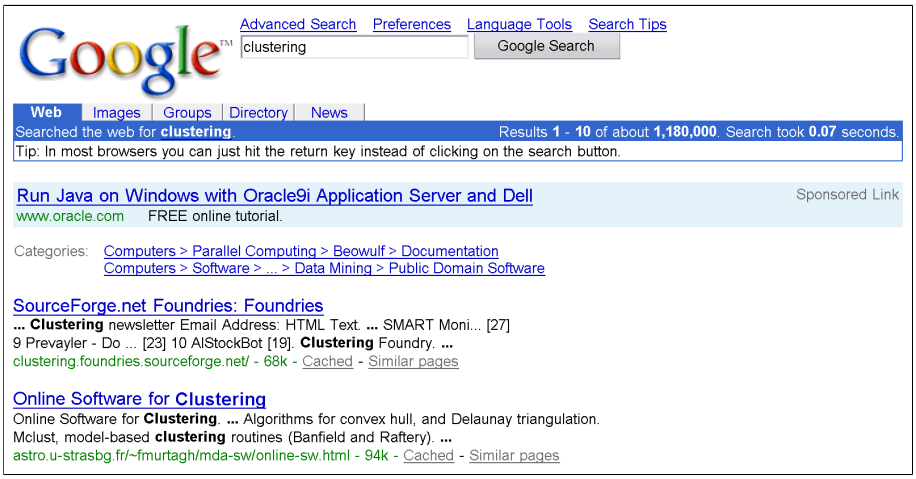
Below we give an outline of the most common search results presentation interfaces.

Ranked list presentation

The ranked list is by far the most commonly used presentation interface nowadays. In this model the documents retrieved in response to a query are sorted according to the relevance

to the query – most relevant first. A single entry in the ranked list usually consists of the title of the document, its URL[7] and a short excerpt from it called a snippet.

Figure 2.5 An example ranked list presentation interface



Popular and widely supported though the ranked list presentation is, it has a number of serious disadvantages [Zamir and Etzioni, 99] [Weiss, 01]:

a. Users must sift through a long list of documents, some of which are irrelevant
b. The reason a document was included in the results is not explicit
c. The relation of a document to the query is not explicit
d. No explicit information is provided about the relationships between documents on the list
e. All documents on the list must be sorted even though some of them may not relate to each other and are thus not comparable

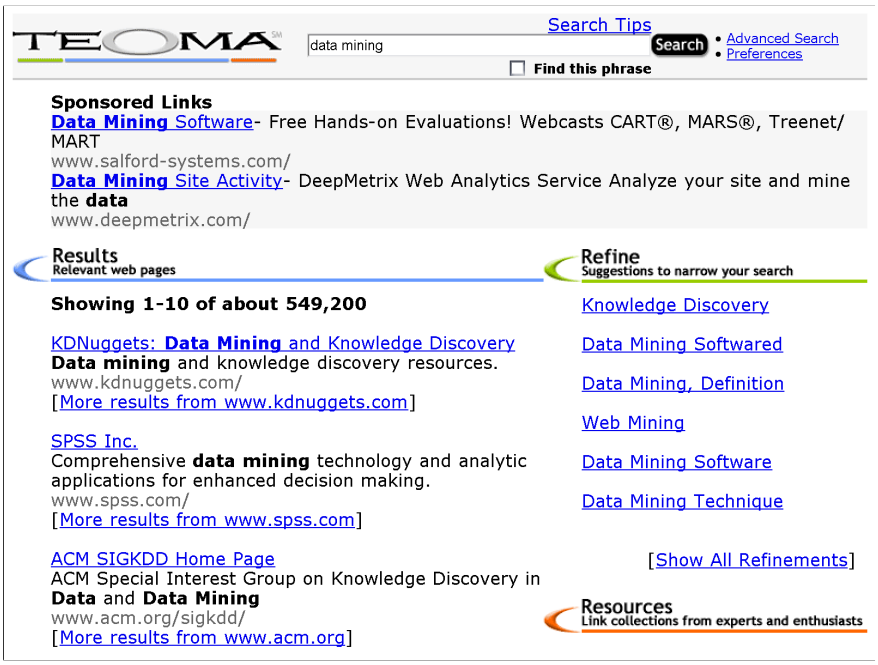
Relevance feedback

One of the techniques meant to improve the ranked list presentation interface is relevance feedback. Relevance feedback is a process where a search engine suggests different ways in which the user's query could be expanded in order to produce more focused results. The majority of relevance feedback solutions are based on the analysis of the text of the query and the documents it returned, AltaVista Prisma [AltaVista Prisma, 03] being an example of such a solution. Teoma [Teoma, 03a] however, uses a different approach:

> *To determine the authority and thus the overall quality and relevance of a site's content, Teoma uses Subject-Specific Popularity. Subject-Specific Popularity ranks a site based on the number of same-subject pages that reference it, not just general popularity. [...] Like social networks in the real world, the Web is clustered into local communities. Communities are groups of Web pages that are about or are closely related to the same subject. [...] These communities are presented under the heading "Refine" on the results page.* [Teoma, 03b]

[7] Uniform Resource Locator

Figure 2.6
Teoma relevance feedback interface



Human-made web directories

Web directories are created in the process of manual assignment of Internet resources to the branches of a hierarchical thematic catalogue. Such a strategy guarantees that the directory is free from senseless documents, but on the other hand, due to the low speed of updating, some of its contents may be outdated. Moreover, web directories cover only a tiny fraction of the Web and hence they lack specific categories.

The three most popular web directories are now Yahoo [Yahoo, 03], LookSmart [LookSmart, 03] and Open Directory Project [ODP, 03]:

Figure 2.7
Open Directory Project main categories

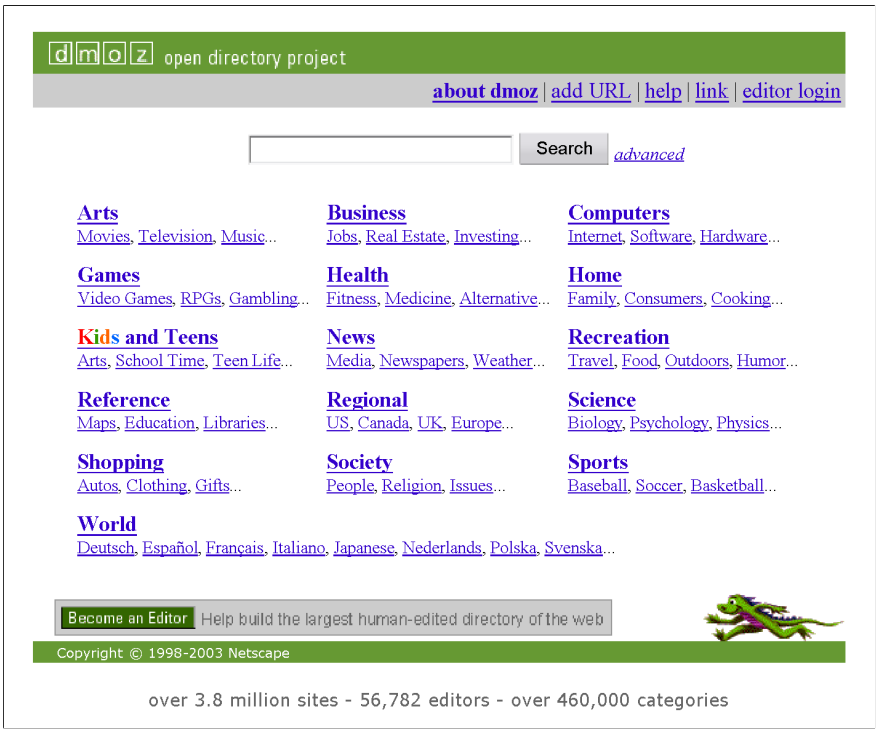


Table 2.1
Web Directory Sizes [Sullivan, 03]

Service	Editors	Categories	Links	As of
ODP	56,782	460,000	3.8 million	5/03
LookSmart	200	200,000	2.5 million	8/01
Yahoo	100+	n/a	1.5 to 1.8 million	8/00

Although human-made web directories are not exactly search results presentation interfaces, we have included them here as they still serve as a valuable source of information for people searching the Internet [Zamir and Etzioni, 99].

Clustering interfaces

Clustering of web search results is an attempt to organise the results into a number of thematic groups in the manner a web directory does it. This approach, however, differs from the human-made directories in many aspects. First of all, only documents that match the query are considered while building the topical groups. Clustering is thus preformed *after* the documents matching the query are identified. Consequently, the set of thematic categories is not fixed – they are created dynamically depending on the actual documents found in the results. Secondly, as the clustering interface is part of a search engine, the assignment of documents to groups must be done efficiently and on-line. For this reason it is unacceptable to download the full text of each document from the Web – clustering ought to be performed based solely on the snippets returned by the search service.

Clustering interfaces employ a fairly new class of algorithms called **post-retrieval document clustering algorithms**. In [Zamir and Etzioni, 98] requirements are given that such algorithms must fulfil:

Post-retrieval document clustering algorithm requirements [Zamir and Etzioni, 98]

a. **Relevance**

The algorithm ought to produce clusters that group documents relevant to the user's query separately from irrelevant ones.

b. **Browsable Summaries**

The user needs to determine at a glance whether a cluster's contents are of interest. We do not want to replace sifting through ranked lists with sifting through clusters. Therefore the algorithm has to provide concise and accurate descriptions of the clusters.

c. **Overlapping clusters**

Since documents have multiple topics, it is important to avoid confining each document to only one cluster [Hearst, 98].

d. **Snippet tolerance**

The algorithm ought to produce high quality clusters even when it only has access to the snippets returned by the search engines, as most users are unwilling to wait while the system downloads the original documents off the Web.

e. **Speed**

As the algorithm will be used as part of an on-line system, it is crucial that it does not introduce noticeable delay to the query processing. Clustering aims at allowing the user to browse through at least an order of magnitude more documents compared to a ranked list.

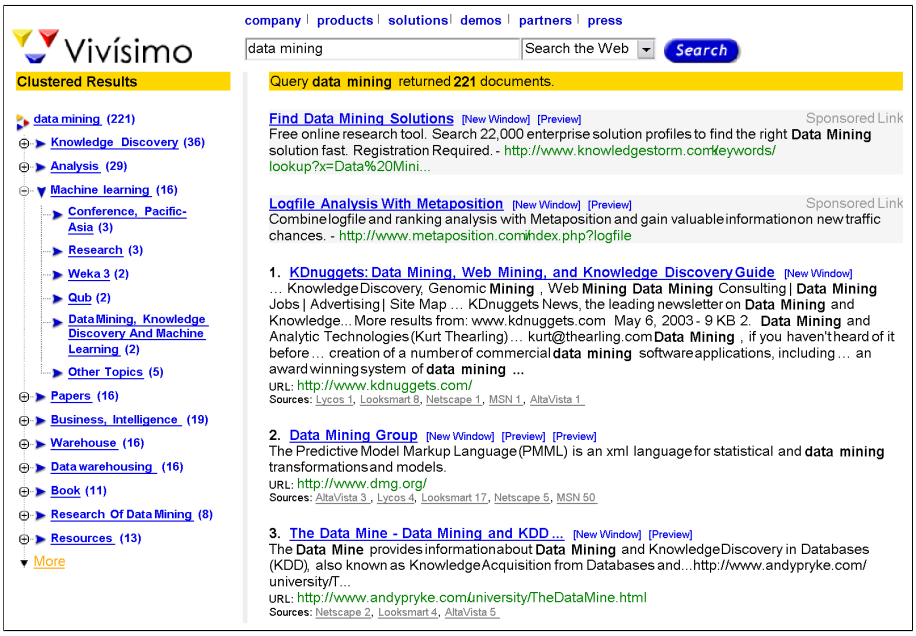
f. **Incremental processing**

To save time, the algorithm should start to process each snippet as soon as it is received over the Web.

Historically, Gather/Scatter [Hearst and Pedersen, 96] was the first practical implementation of the post-retrieval search results clustering idea. Following was Grouper [Zamir and Etzioni, 99] in which Suffix Tree Clustering (STC) – a novel phrase-based document clustering algorithm was introduced. Presently, neither of the two systems is publicly available. Other approaches to search results clustering are Class Hierarchy Construction Algorithm [Shenker et al., 01] and Semantic On-line Hierarchical Clustering [Zhang and Dong, 01].

Of a handful of the contemporary web search clustering interfaces Vivisimo seems to be most mature. Visisimo is a meta-search engine that organises search results into a hierarchical structure of thematic groups. Noteworthy is the excellent quality of the clusters' descriptions:

Figure 2.8 *Vivisimo clustering search engine*



Other examples of similar interfaces are iBoogie [iBoogie, 03], WiseNut [WiseNut, 03] and KillerInfo [KillerInfo, 03]

2.2 Techniques of Automatic text processing

The process of searching a collection of documents[8] involves a number of text processing techniques. Some of the techniques, such as the lexical analysis, are essential parts of the process, whereas other, such as the natural language processing, are optional and are meant to increase the quality of search results. Below, we review the most common of the text processing operations and analyse their applications to the Web Search Problem.

2.2.1 Lexical Analysis

Lexical analysis is a process of converting a stream of characters (the text of a document) into a stream of words. Thus discovered words will be used in further processing, e.g. in building an inverted index of the document collection. In the simplest case, the lexical analyser could recognise white spaces[9] as word separators. In practical implementations however, more sophisticated analysis is required including special treatment of such characters as digits, hyphens and punctuation marks [Baeza-Yates et al., 99]. Numbers, for example, are usually bad index terms as without the surrounding context they carry little meaning. It is thus a common practice to omit them in the process of building an index. Digits on the other hand, especially when used in acronyms (e.g. J2EE[10]) are intrinsic parts of index terms and should not be discarded.

Web-specific lexical analysis

In the World Wide Web environment, during the lexical analysis, additional Internet-specific features of a text should be taken into account. Special characters, such as "@", ought to be analysed within their surrounding context so that e.g. e-mail addresses are not split into separate words. HTML tags, although they have no conceptual meaning and should be disregarded as index terms, can help the lexical analyser identify specific functions of particular words – terms appearing in, for example, the title of a document can be given special treatment in further processing [Riboni, 02].

The overall effectiveness of the majority of text processing algorithms, including web searching, greatly depends on the quality of the underlying lexical analysis. Fortunately, a variety of tools are available (e.g. Regular Expressions, lexical analyser generators) that allow efficient implementation of even very complex lexical processing strategies.

2.2.2 Elimination of stopwords

In every collection of documents a few words exist that recur exceptionally frequently. Words that occur in over 80% of the documents have very limited discrimination power and are practically useless in the process of searching the collection. Such words are referred

---

[8] In this context the term "process" refers both to the classic Information Retrieval and to the Web Search Problem

[9] Characters that represent space between words in the text, such as: line feed, carriage return, tabulation and ordinary spaces

[10] Java 2 Enterprise Edition

to as **stopwords** or **stopterms** and are normally excluded from the set of index terms. Natural candidates for stopwords are articles (e.g. "*the*"), prepositions (e.g. "*for*", "*of*") and pronouns (e.g. "*I*", "*his*").

In practical implementations, during stopword elimination, lists of frequent words are used called **stoplists**. While for the English language such lists can be easily obtained from the Internet, in case of other languages it is much more difficult. An example set of common terms for Polish can be found in [Kurcz et al., 90]. In [Weiss, 01], article archives of a Polish daily newspaper *Rzeczpospolita* were used to build a stoplist for the Polish language.

Internet search engines can employ stoplists not only to increase the precision of searching but also to reduce the size of the document index. Applying stopword elimination can result in a decrease of 40% or more in the size of an inverted index [Baeza-Yates et al., 99]. Very often however, such index compression is achieved at the cost of reduced recall – with strong common term filtering documents containing a phrase "*to be or not to be*" may turn out impossible to find.

2.2.3 Stemming

Frequently, users specify a word in a query, but only variants of that word are present in the document collection. Plurals, gerunds and past forms are examples of grammatical variations that may prevent a match between a query and a document. A partial solution to this problem is replacing all words with their respective stems.

A **stem** is a portion of a word that is left after removal of its affixes (i.e. suffixes and prefixes). Thus, with the use of a **stemmer** (short for a **stemming algorithm**) different grammatical forms of a word can be reduced to one base form. For example, the words: *connected*, *connecting*, *interconnection* should be transformed to the word *connect*.

Non-English stemming

For the English language a variety of stemming algorithms are available, Porter stemmer [Porter, 80] being the most commonly used. For other languages however, as it is the case with stoplists, the availability of free of charge algorithms is limited. In [Weiss, 01] issues of stemming in Polish are discussed and a design of a quasi-stemmer for that language is proposed. A free of charge Java implementation of a Polish stemming engine can be obtained from [Weiss, 03b].

In the World Wide Web environment stemming is used as a means for improving retrieval performance. With the reduction of different variants of the same root word, a wider range of documents can be returned in response to a query and hence better recall can be achieved. Furthermore, as a side effect, stemming may decrease the size of the document index as it reduces the number of distinct index terms.

2.2.4 Spell checking

When formulating their queries, not only do the users use different grammatical forms of words, but also tend to use different spelling to describe the same concept. Whereas the

most common reason for that is probably a spelling mistake, some of the users simply *do not know* how to spell certain words:

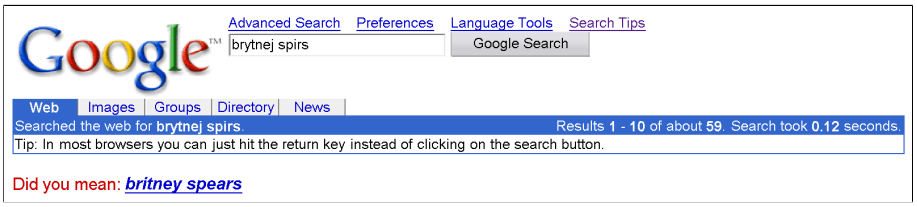


Figure 2.9 *Google's spell checking feature*

As only a minority of all documents on the Web contain incorrectly spelled words, misspelled queries severely limit the recall. To solve the problem, many search engines try to suggest the correct spelling of the user's query. An interesting example [Google, 03b] of the efficiency of Google's spell checker is a list of 593 different ways in which the query "*Britney Spears*" was initially spelled and then transformed to the correct form.

2.2.5 Language identification

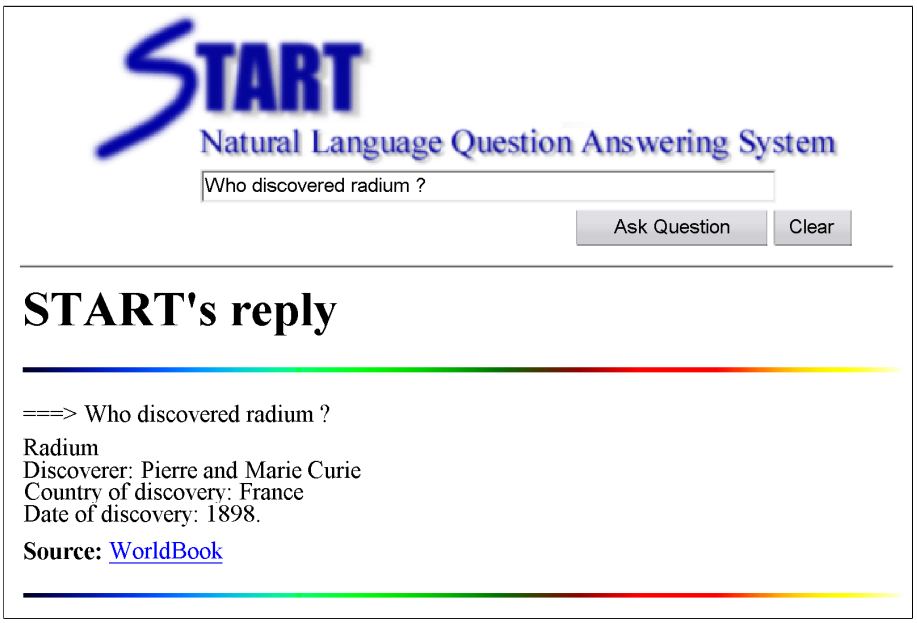
Stopwords elimination, stemming and spell checking techniques make an implicit assumption that the language of the processed text is known. While it may be the case with relatively small collections of documents, in general not only does the language remain unknown, but also it is rarely stated explicitly in the text. To make matters even worse, large document collections such as the World Wide Web, are actually mixtures of texts written in tens of different languages. Clearly, algorithms are needed that will enable the automatic identification of the language of a text.

In [Grefenstette, 95] two statistical methods of automatic language identification are compared. The **trigram** technique works on the premise that, for example, a word ending with *-ck* is more likely to be an English word than a French word, and similarly a word ending with *-ez* is more likely to be French. The intuition behind the **small word** technique is that stopwords (e.g. prepositions or conjunctions) appear in almost all texts and hence are good clues for guessing the language. On texts longer than 30 words both methods perform equally well, the identification error being less than 0.5%. On texts shorter than 5 words however, the trigram-based algorithm proves superior with the average error rate less than 7%. This shows that effective language identification is possible even on bare document titles or web snippets.

2.2.6 Natural Language Processing

Some searchers when looking for information on the Web, instead of using synthetic queries would prefer asking *real* questions. Such type of search requires finding the documents that provide *an answer* to the question rather than documents that just happen to contain certain words. Ideally, the search engine would not even return the documents but the sole answer:

Figure 2.10
START – Natural Language Question Answering System



Real questions, expressed by means of **natural language**, are difficult to process as they require the computer system to understand the intent of the question. This in turn involves complex analysis of syntactic and semantic relationships between words and phrases found in the question. Natural Language Processing (NLP) is a sub-area of Artificial Intelligence that addresses such problems. For the time being however, very few publicly available search engines use NLP algorithms to aid the process of finding information on the Web. Examples of experimental interfaces that accept queries expressed in natural language are AnswerBus [AnswerBus, 03] and START [START, 03].

2.3 Text clustering algorithms

Clustering is a process of forming groups (clusters) of similar objects from a given set of inputs. Good clusters have the characteristic that objects belonging to the same cluster are "similar" to each other, while objects from two different clusters are "dissimilar". The idea of clustering originates from statistics where it was applied to numerical data. However, computer science and data mining in particular, extended the notion to other types of data such as text or multimedia.

Clearly, web snippets belong to the class of textual data and hence it is the text clustering algorithms that should be regarded as a base for the web search results clustering systems. Therefore, below we give an outline of three distinct approaches to the problem of text clustering.

2.3.1 Classic approach

The classic approach attempts to adopt the well-known clustering algorithms, originally designed for numerical data, such as Hierarchical Agglomerative Clustering (HAC) or K-

means, to the data of textual type. The algorithms require that for every two objects[11] in the input collection a **similarity measure** be defined. The measure, which is usually calculated as a single numerical value, represents the "distance" between these objects – objects that are "close" to each other in this sense will be placed in the same cluster. Thus, to use the classic algorithms for the new type of data, the measure of similarity between two texts must be defined. Many different approaches to this problem have been proposed [Baeza-Yates et al., 99]:

a. **Hamming distance**

The Hamming measure requires that the input texts be of the same length. The distance between such texts is calculated as the number of positions that have different characters. Thus, the distance is 0 when the texts are equal.

b. **Edit distance**

The Edit distance, often called the Levenstein distance, is defined as the minimum number of character insertions, deletions and substitutions that must be performed on any of the texts to make them equal. For instance, the edit distance between "*color*" and "*colour*" is one, while the edit distance between "*surgery*" and "*survey*" is two.

c. **Cosine distance**

The cosine measure is based on the Vector Space Model. The model assumes that every text in the collection is represented by a multidimensional vector and the distance between two texts is calculated as a dot product of appropriate vectors. For a detailed description of the Vector Space Model refer to Chapter 3.

After the the distance measure between two texts has been chosen, classic clustering algorithms are ready be used on textual data. Below we briefly describe the main ideas behind the two most popular classic approaches: Hierarchic Agglomerative Clustering and K-means. For more details regarding the algorithms refer to [Salton, 89] and [Weiss, 01].

Agglomerative Hierarchical Clustering

In each step of the Agglomerative Hierarchical Clustering (HAC) algorithm, two objects, an object and a cluster or two clusters that are closest to each other are merged into a new group. In this way the relationships between input objects are represented in a tree-like **dendrogram**. In [Wroblewski, 03] applications of HAC to web search results clustering are discussed; several strategies of pruning of the dendrogram are also proposed.

K-means

K-means is an iterative algorithm in which clusters are built around k central points[12] called **centroids**. The algorithm starts with a random set of centroids and assigns each object to its closest centroid. Then, repeatedly, for each group, based on its members, a new central point (new centroid) is calculated and object assignments to their closest centroids are

[11] The objects are usually described by sets of numerical attributes

[12] k is a parameter of the algorithm and must be known before the clustering starts.

changed if necessary. The algorithm finishes when no object reassignments are needed or when certain amount of time elapses.

2.3.2 Suffix Tree Clustering

The Suffix Tree Clustering (STC) algorithm groups the input texts according to the identical *phrases* they share [Zamir, 99]. The rationale behind such approach is that phrases, compared to single keywords, have greater descriptive power. This results from their ability to retain the relationships of proximity and order between words. A great advantage of STC is that phrases are used both to discover and to describe the resulting groups.

The Suffix Tree Clustering algorithm works in two main phases: **base cluster** discovery phase and base cluster merging phase. In the first phase a **generalised suffix tree** of all texts' sentences is built using words as basic elements. After all sentences are processed, the tree nodes contain information about the documents in which particular phrases appear. Using that information documents that share the same phrase are grouped into base clusters of which only those are retained whose score exceeds a predefined **Minimal Base Cluster Score**. In the second phase of the algorithm, a graph representing relationships between the discovered base clusters is built based on their similarity and on the value of the **Merge Threshold**. Base clusters belonging to coherent subgraphs of that graph are merged into final clusters. The following formulation of the algorithm is given in [Weiss, 01]:

Figure 2.11 *Suffix Tree Clustering pseudo-code*

```
/**
 * Suffix Tree Clustering algorithm
 */

split text into sentences consisting of words;

/* Phase 1a. Creation of a Generalized Suffix Tree of all sentences */
for each document
{
  for each sentence
  {
    if (sentence length > 0)
    {
      insert sentence and all its substrings into generalised
      suffix tree and update internal nodes with the index to current
      document while rearranging the tree;
    }
  }
}

/* Phase 1b. Build a list of base clusters */
for each node in the tree
{
  if (number of documents in node's subtree > 2)
  {
    if (candidateBaseClusterScore > Minimal_Base_Cluster_Score)
    {
      add a base cluster to the list of base clusters;
    }
  }
}

/* Phase 2. Merge base clusters */
build a graph where nodes are base clusters and there is a link between
node A and B if and only if the number of common documents indexed by
A and B is greater than the Merge_Threshold;

clusters are coherent subgraphs of that graph;
```

A detailed example illustrating the STC algorithm along with its evaluation based on standard Information Retrieval metrics and user feedback is presented in [Zamir, 99]. In [Weiss, 01] STC's applicability to clustering web search results in Polish is analysed. The influence of language properties, such as rich inflection, on the results produced by STC is investigated in [Stefanowski and Weiss, 03].

Strong and weak points of STC

A clear advantage of Suffix Tree Clustering is that it uses phrases to provide concise and meaningful descriptions of groups. However, as noted in [Weiss, 01] STC's thresholds play a significant role in the process of cluster formation, and they turn out particularly difficult to tune. Also, STC's phrase pruning heuristic tends to remove longer high-quality phrases, leaving only the less informative and shorter ones. Finally, as pointed out in [Zhang and Dong, 01], if a document does not include any of the extracted phrases it will not be included in the results although it may still be relevant.

2.3.3 Semantic Online Hierarchical Clustering

The Semantic Online Hierarchical Clustering (SHOC) [Zhang and Dong, 01] is a web search results clustering algorithm that was originally designed to process queries in Chinese. Although it is based on a variation of the Vector Space Model called Latent Semantic Indexing (LSI) and uses phrases in the process of clustering, it is much different from the previously presented approaches. To overcome the STC's low quality phrases problem, in SHOC Zhang and Dong introduce two novel concepts: **complete phrases** and a **continuous cluster definition**.

Complete phrases

To avoid extracting partial phrases, which are meaningless in Chinese and very often also in English[13], SHOC introduces a notion of phrase (or substring[14]) completeness. Given a document T of length N, S – the complete substring of T is defined as follows:

The complete substring definition [Zhang and Dong, 01]

> **Definition** — S is a complete substring of T when S occurs in k distinct positions $p_1, p_2, \ldots, p_k$, in T, and the $(p_i\text{-}1)$th character in T is different from the $(p_j\text{-}1)$th character for at least one pair (i, j), $1 \le i < j \le k$ (left-completeness), and the $(p_i + |S|)$th character is different from the $(p_j + |S|)$th character for at least one pair (i, j), $1 \le i < j \le k$ (right-completeness).

To help the Reader understand the above definition, in Figure 2.12 we show an example string of characters in which all non-complete (i.e. partial), left- and right-complete phrases can be found.

Figure 2.12 *Complete phrases example*

x a b c d e a b c d f a b c d g

[13] An example of a partial phrase in English could be "Senator Hillary", which is only a fragment of a much more informative phrase "Senator Hillary Rodham Clinton"

[14] From the definition point of view there is no difference between words and characters as the basic elements of a text

The phrase "bc" is not right-complete as *the same* character "d" directly follows *all* of its occurrences. It is not a left-complete phrase either as the same character "a" directly precedes all of its appearances. However, the phrase "bcd" is right-complete because its subsequent occurrences are followed by *different* letters, "e", "f", and "g", respectively. To make the "bcd" phrase also left-complete it must be extended to the left so that it contains the "a" character. In this way, a complete (i.e. both left- and right-complete) phrase "abcd" has been identified.

In [Zhang and Dong, 01] an algorithm is proposed that uses a data structure called **suffix array** to identify complete phrases in O(n) time, n being the total length of all processed documents. For the details regarding suffix arrays refer to Chapter 3. In Chapter 4 a slightly modified version of Zhang and Dong's algorithm is described in detail.

Continuous cluster definition

In contrast to the Suffix Tree Clustering, SHOC adopts a continuous cluster definition, which allows documents to belong to different clusters with different intensity. In a natural way, such definition meets the requirement of overlapping clusters. Additionally, it provides a method of ordering documents *within* clusters. The continuous cluster definition is the following.

The continuous cluster definition [Zhang and Dong, 01]

> **Definition —** A cluster C_g of m objects t_1, t_2, ..., t_m is defined by an m-dimensional vector (the cluster vector) $\mathbf{x_g}$, $|\mathbf{x_g}|=1$. Each component $\mathbf{x_g}(i)$ of the cluster vector represents the intensity with which t_i belongs to C_g.

In [Zhang and Dong, 01] a proof is given that in order for a set of continuous clusters to be meaningful, all cluster vectors must be pairwise orthogonal. Additionally, Zhang and Dong prove that clusters that meet such requirement can be found with the use of the Singular Value Decomposition. For details and examples on Singular Value Decomposition refer to Chapter 3.

The SHOC algorithm

The SHOC algorithm works in three main phases: complete phrase discovery phase, base cluster discovery phase and cluster merging phase. In the first phase, suffix arrays are used to discover complete phrases and their frequencies in the input collection. In the second phase, using Singular Value Decomposition a set of orthogonal base clusters is obtained. Finally, in the last phase, base clusters are merged into a hierarchical structure. In Figure 2.13 a pseudo-code formulation of the SHOC algorithm is given.

***Figure 2.13** Semantic Hierarchical Online Clustering pseudo-code*

```
/**
 * Semantic On-line Hierarchical Clustering algorithm
 */

split text into sentences consisting of words;

/* Phase 1. Discover complete phrases */
build a suffix array of the text;
while scanning the suffix array identify complete phrases;

/* Phase 2. Discover base clusters */
build a phrase-document matrix;
```

```
perform the Singular Value Decomposition of the matrix;
the right singular vectors of the matrix are the base cluster vectors;

/* Phase 3. Build cluster hierarchy */
let X and Y be two sets representing base clusters;

/**
 * sum(X, Y) denotes the sum of sets X and Y
 * intersection(X, Y) denotes the intersection of sets X and Y
 * size(A) denotes the number of elements in set A
 */
if (size(sum(X, Y)) / size(intersect(X, Y)) > Merge_Threshold_1)
{
  merge X and Y into one cluster;
}
else
{
  if (size(X) > size(Y))
  {
    if (size(intersect(X, Y)) / size(Y) > Merge_Threshold_2)
    {
      let Y become X's child;
    }
  }
  else
  {
    if (size(intersect(X, Y)) / size(X) > Merge_Threshold_2)
    {
      let X become Y's child;
    }
  }
}
```

One of the drawbacks of SHOC is that Zhang and Dong provide only vague comments on the values of Merge Threshold 1 and 2 and the method which is used to describe the resulting clusters. Additionally, a test implementation of SHOC, which we prepared during the course of writing this thesis, shows that in many cases the Singular Value Decomposition produces unintuitive (sometimes even close to "random") continuous clusters. The reason for this lies probably the fact that the SVD is performed on document snippets rather than the full texts as it was in its original applications.

The inspiration for LINGO

The above drawbacks of SHOC were the direct incentive for us to create LINGO – our proposed web search results clustering algorithm. We have decided to use a slightly modified version of SHOC's complete phrase discovery algorithm. We also employ the Singular Value Decomposition, but rather than use it to find cluster contents, we utilise this technique to identify meaningful group labels. In Chapter 4, we present an in-depth description of LINGO, pointing out all differences between SHOC and our approach.

3 LINGO - Preliminary Concepts

In this chapter we introduce a number of Information Retrieval concepts that serve as a base for LINGO – our proposed web search results clustering algorithm. We start with explaining the Vector Space Model – a well-established and largely successful Information Retrieval technique. We describe the model with respect to document indexing and query matching. Next, we shift to a more advanced IR concept – the Latent Semantic Indexing. We finish this chapter with a description of suffix arrays – a data structure which facilitates a wide variety of text operations.

The Reader may wish to consult Appendix B for definitions of the linear algebra notions appearing in this chapter. For detailed explanations we refer the Reader to [Nicholson, 86], [Hartfiel and Hobbs, 87] or [Opial, 69].

3.1 Vector Space Model

Various mathematical models have been proposed to represent Information Retrieval systems and procedures. The Boolean model compares Boolean query statements with the term sets used to identify document content. The probabilistic model is based on computation of relevance probabilities for the documents in the collection. Finally, the Vector Space Model uses term sets to represent both queries and documents, employing basic linear algebra operations to calculate global similarities between them. Of the above models the Vector Space Model is the simplest to use and in some ways most productive [Salton, 89].

In the Vector Space Model (VSM), every document in the collection is represented by a multidimensional vector. Each component of the vector reflects a particular key word or term connected with the given document. The value of each component depends on the degree of relationship between its associated term and the respective document. Many schemes for measuring the relationship, very often referred to as **term weighting**, have been proposed. In the following subsection we review the three most popular.

3.1.1 Term weighting

Term weighting is a process of calculating the degree of relationship (or association) between a term and a document. As the Vector Space Model requires that the relationship be described by a single numerical value, let a_{ij} represent the degree of relationship between term i and document j.

Binary weighting In the simplest case the association is binary: a_{ij}=1 when key word i occurs in document j, a_{ij}=0 otherwise. The binary weighting informs about the *fact* that a term is somehow related to a document but carries no information on the *strength* of the relationship.

Term frequency weighting A more advanced term weighting scheme is the **term frequency**. In this scheme

$$a_{ij}=tf_{ij}$$

where tf_{ij} denotes how many times term i occurs in document j. Clearly, the term frequency is more informative than the simple binary weighting. Nevertheless, it still has some shortcomings. To identify the problem, let us assume that in 95% of the documents in the collection the word "*apple*" occurs frequently. Locally, for each document separately, the word is an excellent indicator of the document's content. However, as the term appears in almost every document in the collection, it does not help distinguish documents from each other. Thus, *globally* the term "*apple*" is less valuable. The drawback of the term frequency scheme is that it focuses only on the local word occurrences.

Tf-idf weighting The ***tf-idf*** (term frequency inverse document frequency) scheme aims at balancing the local and the global term occurrences in the documents. In this scheme

$$a_{ij}=tf_{ij}\cdot\log(N/df_i)$$

where tf_{ij} is the term frequency, df_i denotes the number of documents in which term i appears, and N represents the total number of documents in the collection. The $\log(N/df_i)$, which is very often referred to as the *idf* (inverse document frequency) factor, accounts for the global weighting of term i. Indeed, when a term appears in all documents in the collection, $df_i=N$ and thus the balanced term weight is 0, indicating that the term is useless as a document discriminator. For more details regarding the above term weighting schemes refer to [Salton, 89].

3.1.2 Query matching

The term-document matrix In the Vector Space Model, a collection of d documents described by t terms can be represented as a $t\times d$ matrix A, hereafter referred to as the **term-document matrix**. Each element a_{ij} of the term-document matrix represents the degree of relationship between term i and document j by means of one of the presented term weighting schemes. The column vectors of A, called **document vectors**, model the documents present in the collection, while the row vectors of A, called **term vectors**, represent the terms used in the process of indexing the collection. Thus, the document vectors *span* the document collection, which means they contain the whole semantic content of the collection. What should be stressed here is that not every vector in the column space of A represents a *real* document – a vector being a linear combination of two real document vectors, for example, may turn out to be meaningless. However, it is the geometric relationships between document vectors that are important when modelling the similarity between a document and a query.

In the Vector Space Model, a user query is represented by a vector in the column space of the term-document matrix. This means that the query can be treated as a pseudo-document

that is built solely of the query terms. Therefore, in the process of query matching, documents must be selected whose vectors are geometrically closest to the query vector. A common measure of similarity between two vectors is the cosine of the angle between them. In a $t \times d$ term-document matrix A, the cosine between document vector a_j and the query vector q can be computed according to the formula:

Figure 3.1 The cosine distance formula

$$\cos\theta_j = \frac{a_j^T q}{\| a_j \| \| q \|} = \frac{\sum_{i=1}^{t} a_{ij} q_i}{\sqrt{\sum_{i=1}^{t} a_{ij}^2} \sqrt{\sum_{i=1}^{t} q_i^2}}$$

where a_j is the jth document vector, t is the number of terms and $||a||$ denotes the length of vector a. As multiplying either the a_j vector or the q vector by a constant does not change the cosine, the vectors can be scaled by any convenient value. Very often, both document vectors and the query vector are normalised to a unit length. In this way a single vector-matrix multiplication $q^T A$ yields a vector of similarities between the query and each document in the collection. Documents whose similarity to the query exceeds a predefined threshold (e.g. 0.8) are returned as the search result.

3.1.3 An example

Figure 3.2 demonstrates how from a collection of seven document titles a term-document matrix can be created. For the sake of clarity, we consider only those terms that appear in the documents more than once and are not stopwords. When building the term-document matrix the term frequency weighting scheme was used.

Figure 3.2 Vector Space Model example – input data

The t=5 **terms**:

T1: Information
T2: Singular
T3: Value
T4: Computations
T5: Retrieval

The d=7 **documents**:

D1: Large Scale Singular Value Computations
D2: Software Library for the Sparse Singular Value Decomposition
D3: Introduction to Modern Information Retrieval
D4: Using Linear Algebra for Intelligent Information Retrieval
D5: Matrix Computations
D6: Singular Value Analysis of Cryptograms
D7: Automatic Information Organization

The 5x7 **term-document** matrix before column length normalization:

$$A = \begin{pmatrix} 0.00 & 0.00 & 1.00 & 1.00 & 0.00 & 0.00 & 1.00 \\ 1.00 & 1.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \\ 1.00 & 1.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \\ 1.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.00 & 1.00 & 0.00 & 0.00 & 0.00 \end{pmatrix}$$

The 5x7 **term-document** matrix after column length normalization:

$$\hat{A} = \begin{pmatrix} 0.00 & 0.00 & 0.71 & 0.71 & 0.00 & 0.00 & 1.00 \\ 0.58 & 0.71 & 0.00 & 0.00 & 0.00 & 0.71 & 0.00 \\ 0.58 & 0.71 & 0.00 & 0.00 & 0.00 & 0.71 & 0.00 \\ 0.58 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.71 & 0.71 & 0.00 & 0.00 & 0.00 \end{pmatrix}$$

In Figure 3.3 an example of the query matching process for a query *Singular Value* is shown. Assuming 0.8 as the value of the similarity threshold, D1, D2 and D6 will be returned as the search result. Other documents, which do not concern the topic, are correctly ignored.

Figure 3.3 *Vector Space Model example – query matching*

The query: ***Singular Value***

The **query vector** before length normalization:

$$q = (0.00 \quad 1.00 \quad 1.00 \quad 0.00 \quad 0.00)^T$$

The **query vector** after length normalization:

$$\hat{q} = (0.00 \quad 0.71 \quad 0.71 \quad 0.00 \quad 0.00)^T$$

The query-document **similarity vector:**

$$\hat{q}^T A = (0.83 \quad 1.00 \quad 0.00 \quad 0.00 \quad 0.00 \quad 1.00 \quad 0.00)^T$$

The **matched documents**:

D1: Large Scale Singular Value Computations
D2: Software Library for the Sparse Singular Value Decomposition
D6: Singular Value Analysis of Cryptograms

3.2 Latent Semantic Indexing

Successful though the classic Vector Space Model is, it does not address many of the problems of modern Information Retrieval. First of all, the model is based on literal matching between terms from a document and those of a query. Due to the phenomenon of **synonymy** however, there are usually many ways in which the same idea can be expressed – the terms found in the user's query may not match those of a relevant document. In this way, some of the documents will not appear in the search results even though they should. Second of all, the Vector Space Model model makes an assumption that the keywords used to index the document collection are mutually independent. Nevertheless, in real applications it may not always be the case. For this reason significant volumes of redundant information are present in the classic term-document matrix. Last but not least, the classic VSM model uses single words to describe documents while it is the *concepts* that the majority of users look for.

Latent Semantic Indexing (LSI) is a novel Information Retrieval technique that was designed to address the deficiencies of the classic VSM model:

> *LSI tries to overcome the problems of lexical matching by using statistically derived conceptual indices instead of individual words for retrieval. LSI assumes that there is some underlying or latent structure in word usage that is partially obscured by variability in word choice. A truncated Singular Value Decomposition (SVD) is used to estimate the structure in word usage across documents. Retrieval is then performed using a database of singular values and vectors obtained from the*

truncated SVD. Performance data shows that these statistically derived vectors are more robust indicators of meaning than individual terms. [Berry et al., 95]

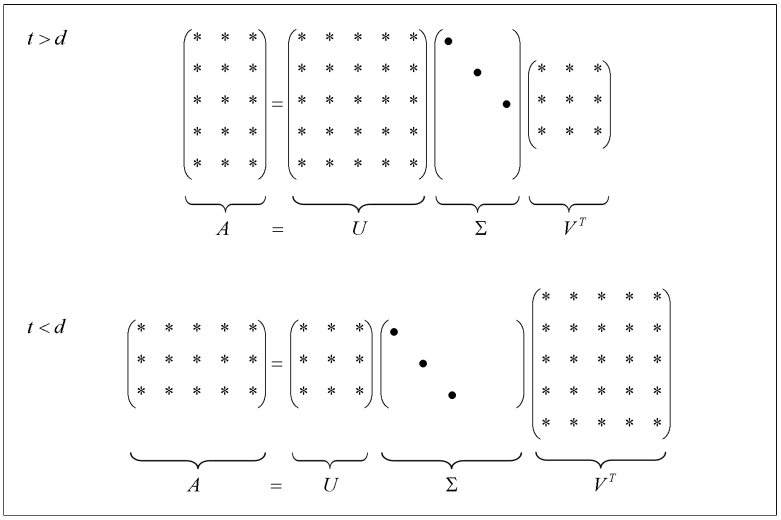
Applications of Latent Semantic Indexing

Applications of the Latent Semantic Indexing are diverse and vary from classic Information Retrieval tasks to psychological modelling. In [Littman et al., 96] LSI was used to implement a cross-language document retrieval system for a French-English collection. In response to a query formulated in either of the languages, the system returns relevant texts written both in English and in French. The authors provide evidence that LSI-based cross-language retrieval performs comparably with methods that employ machine translation techniques. Latent Semantic Indexing has also been used to model some of the associative relationships observed in human memory [Landauer and Dumais, 96]. Landauer and Dumais tested how well an LSI space would mimic the knowledge needed to pass the TOEFL[15] synonym test. LSI scored 64% correct, an average TOEFL student achieving exactly the same result. More examples of, sometimes unexpected, applications of Latent Semantic Indexing the Reader can find in [Berry et al., 95] and [Landauer et al., 98].

3.2.1 Singular Value Decomposition

The fundamental mathematical construct underlying the LSI is the Singular Value Decomposition of the term-document matrix. The decomposition breaks a $t \times d$ matrix A into three matrices U, Σ and V such that $A=U\Sigma V^T$. In figure Figure 3.4 the relative sizes of the three matrices are shown when $t>d$ and when $t<d$.

Figure 3.4 *Singular Value Decomposition result matrices*



U is the $t \times t$ orthogonal matrix whose column vectors are called the left singular vectors of A, V is the $d \times d$ orthogonal matrix whose column vectors are termed the right singular vectors of A, and Σ is the $t \times d$ diagonal matrix having the singular values of A ordered decreasingly ($\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_{min(t,d)}$) along its diagonal. The rank r_A of the matrix A is equal to the number of its non-zero singular values. The first r_A columns of U form an orthogonal

[15] Test Of English as a Foreign Language

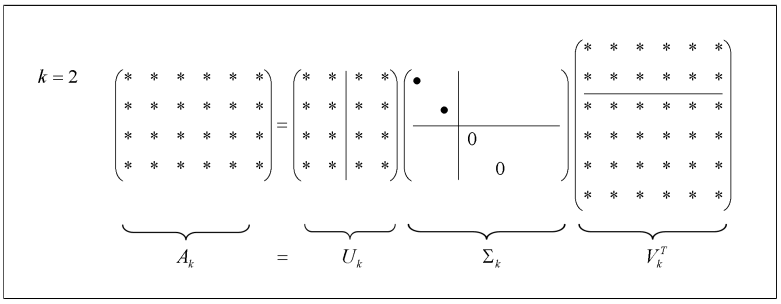
basis for the column space of A. The latter is the central fact exploited by LINGO – refer to Chapter 4 for the details.

3.2.2 Query matching

Contrary to the classic Vector Space Model, query matching in LSI does not employ the original term-document matrix. Instead, a **low-rank approximation** of the term-document matrix is used. As argued in [Berry et al., 95], the rank reduction, which lowers the number of dimensions in which the documents are represented, eliminates noise and extraneous information from the term-document matrix.

Using the Singular Value Decomposition a k-rank approximation A_k of matrix A can be calculated that is closest[16] to the original matrix for a given value of k. The A_k matrix can be obtained from the SVD-derived matrices by setting all but the k largest singular values in the Σ matrix to 0. Thus, $A_k=U_k\Sigma_k V_k^T$, where U_k is the $k\times t$ matrix whose columns are first k columns of U, V_k is the $d\times k$ matrix whose columns are the first k columns of U, and Σ_k is the $k\times k$ matrix whose diagonal elements are the k largest singular values of A (see Figure 3.5).

Figure 3.5
k-rank matrix approximation



Having the k-rank approximation A_k of the term-document matrix, the query matching can be performed as in the classic Vector Space Model – by calculating the cosine of the angle between the query vector and the column vectors of the A_k matrix. If we define e_j to be the jth canonical vector[17] of dimension d, the jth column of A_k is given by $A_k e_j$. Thus, the cosine between the query vector q and the jth document vector can be calculated as follows:

$$\cos\theta_j = \frac{(A_k e_j)^T q}{\| A_k e_j \|_2 \| q \|_2} = \frac{(U_k \Sigma_k V_k^T e_j)^T q}{\| U_k \Sigma_k V_k^T e_j \|_2 \| q \|_2} = \frac{e_j^T V_k \Sigma_k (U_k^T q)}{\| \Sigma_k V_k^T e_j \|_2 \| q \|_2}$$

If we define the vector $s_j=\Sigma_k V_k^T e_j$, the formula reduces to:

$$\cos\theta_j = \frac{s_j^T (U_k^T q)}{\| s_j \|_2 \| q \|_2}$$

and the cosine can be computed without explicitly forming the A_k matrix.

---

[16] The A_k matrix is closest to the original matrix in terms of the Frobenius norm. For details regarding the norm refer to Appendix B.

[17] The jth column of a $d\times d$ identity matrix

Determining the optimal value for k

Interesting is the problem of choosing the optimal value for k. If the value is set too low, some of the information present in the indexed documents may be lost due to the excessive reduction in the number of dimensions. Setting the value too high, on the other hand, may reduce the effect of noise elimination. In [Berry et al., 99] the Frobenius norm of the A and A_k matrices is used to calculate the percentage distance between the original term-document matrix and its approximation. The value of k is then set to such value that the k-rank approximation retains a certain percentage (e.g. 80%) of the original information. Another approach calculates differences between subsequent singular values of the term-document matrix. The value of k at which the largest difference is encountered is taken for further processing [Lerman, 99]. Another approach is to determine k using the Minimum Description Length principle [Zha, 98]

3.2.3 An example

To illustrate the Latent Semantic Indexing technique let us revisit the Vector Space Model example (see Section 3.1.3). In Figure 3.6 results of the SVD decomposition of the length-normalised term-document matrix are shown.

Figure 3.6 *Latent Semantic Indexing example – Singular Value Decomposition results*

$$U = \begin{pmatrix} 0.00 & 0.85 & 0.00 & -0.53 & 0.00 \\ 0.67 & 0.00 & -0.21 & 0.00 & 0.71 \\ 0.67 & 0.00 & -0.21 & 0.00 & -0.71 \\ 0.30 & 0.00 & 0.95 & 0.00 & 0.00 \\ 0.00 & 0.53 & 0.00 & 0.85 & 0.00 \end{pmatrix} \quad \Sigma = \begin{pmatrix} 1.68 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 1.62 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.09 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.62 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix}$$

$$V = \begin{pmatrix} 0.57 & 0.00 & 0.28 & 0.00 & 0.34 \\ 0.57 & 0.00 & -0.28 & 0.00 & 0.48 \\ 0.00 & 0.60 & 0.00 & 0.37 & -0.15 \\ 0.00 & 0.60 & 0.00 & 0.37 & 0.15 \\ 0.18 & 0.00 & 0.88 & 0.55 & -0.20 \\ 0.57 & 0.00 & -0.28 & 0.00 & -0.76 \\ 0.00 & 0.53 & 0.00 & -0.85 & 0.00 \end{pmatrix}$$

Abstract concepts

As noted previously, the column vectors of the U matrix form an orthogonal basis for the column space of the original term-document matrix. Intuitively, as the basis vectors are also expressed in the same space, they can be treated as pseudo-documents that in a way summarise the most important **abstract concepts** present in the collection. The first column vector of the U matrix represents the abstract concept "*Singular Value Computations*" while the second vector is a pseudo-document that deals with "*Information Retrieval*". Indeed, these are the two main ideas around which the documents revolve. In Chapter 4 we show how the basis of the column space of the term-document matrix can be utilised in a clustering algorithm.

Figure 3.7 presents the 2-rank approximation A_2 of the term-document matrix. Due to the significant reduction in the number of dimensions documents 1, 2, 5, 6 and 3, 4, 7 became indistinguishable.

Figure 3.7
Latent Semantic Indexing example – 2-rank approximation

$$A_2 = \begin{pmatrix} 0.00 & 0.00 & 0.85 & 0.85 & 0.00 & 0.00 & 0.85 \\ 0.67 & 0.67 & 0.00 & 0.00 & 0.67 & 0.67 & 0.00 \\ 0.67 & 0.67 & 0.00 & 0.00 & 0.67 & 0.67 & 0.00 \\ 0.30 & 0.30 & 0.00 & 0.00 & 0.30 & 0.30 & 0.00 \\ 0.00 & 0.00 & 0.53 & 0.53 & 0.00 & 0.00 & 0.53 \end{pmatrix}$$

In Figure 3.8 results of a query *Singular Value* are shown. In comparison to the results obtained from the Vector Space Model, one more document – D5 – is returned. Although the document does not contain the exact words used in the query, it is similar to the other retrieved document as they also treat about matrix computations. This is an example of how LSI can address the problems of synonymy.

Figure 3.8
Latent Semantic Indexing example – query matching

The query: ***Singular Value***

The **query vector** before length normalization:

$$q = (0.00 \quad 1.00 \quad 1.00 \quad 0.00 \quad 0.00)^T$$

The **query vector** after length normalization:

$$\hat{q} = (0.00 \quad 0.71 \quad 0.71 \quad 0.00 \quad 0.00)^T$$

The query-document **similarity vector:**

$$\hat{q}^T A_2 = (0.95 \quad 0.95 \quad 0.00 \quad 0.00 \quad 0.95 \quad 0.95 \quad 0.00)^T$$

The **matched documents**:

D1: Large Scale Singular Value Computations
D2: Software Library for the Sparse Singular Value Decomposition
D5: Matrix Computations
D6: Singular Value Analysis of Cryptograms

3.3 Suffix Arrays

A **suffix array** is an alphabetically ordered array of all **suffixes**[18] of a string. Due to the ordering, binary searching[19] of the array is possible and hence string searches of the type "*Is X a substring of T?*" can be answered in O(*P* + log*N*) time, where *P* is the length of *X* and *N* is the length of *T* [Manber and Myers, 94].

Applications of suffix arrays

Suffix arrays are used as a building block in a wide variety of applications. In [Baeza-Yates et al., 99] the data structure is considered as a replacement for the inverted index scheme allowing a wider choice of text searching operations. Another application is the the Burrows-

[18] A suffix is such substring of a string that starts at some position of the string and spans to its last character. More formally, *v* is a suffix of a string *u* if $u=u'v$ for some string u'.

[19] Binary searching is a method of searching a sorted array by repeatedly dividing the search interval in half. Searching an array of *N* elements requires thus O(log*N*) comparisons.

Wheeler compression algorithm [Burrows and Wheeler, 94] in which the time-efficient suffix array construction is crucial for any practical implementation of the algorithm. Suffix arrays have also been employed by two modern web search results clustering algorithms [Zamir, 99][20] [Zhang and Dong, 01] to facilitate the process of frequent phrase identification. Finally, suffix arrays, at the cost of longer construction time, can be a less space-consuming alternative to suffix trees. In practical implementations the space overhead introduced by suffix trees varies from 120% up to 240% of the text size, while suffix arrays need less than 40% of the space occupied by the original text [Baeza-Yates et al., 99].

3.3.1 The data structure

In Figure 3.9 an example suffix array for a string "*to_be_or_not_to_be*" is presented. What should be stressed here is that every suffix of a string is represented by a single integer value (typically 4-bytes long) being equal to the position at which the suffix starts (counting from 0). Thus, it is not necessary to store the texts of particular suffixes of the string – it is only for clarity that we have included them in the figure.

Figure 3.9 *An example suffix array*

Suffix Array		Suffix denoted by S[i]
S[0]	15	_be
S[1]	2	_be_or_not_to_be
S[2]	8	_not_to_be
S[3]	5	_or_not_to_be
S[4]	12	_to_be
S[5]	16	be
S[6]	3	be_or_not_to_be
S[7]	17	e
S[8]	4	e_or_not_to_be
S[9]	9	not_to_be
S[10]	14	o_be
S[11]	1	o_be_or_not_to_be
S[12]	6	or_not_to_be
S[13]	10	ot_to_be
S[14]	7	r_not_to_be
S[15]	11	t_to_be
S[16]	13	to_be
S[17]	0	to_be_or_not_to_be

Preliminary definitions

Before we proceed with a description of a substring searching algorithm based on suffix arrays, let us define the following. Let every string be represented by a sequence of n+1 characters $c_0c_1…c_n$ where the first n characters comprise the actual input string and c_n = $ is a unique terminator character. Let us also assume that the $ character has a value below all other characters. Let T be a string of length N for which a suffix array S consisting of N integers has been built. Further, let T_p denote a suffix of T starting at the pth character of T. In this way $T_{S[0]} < T_{S[1]} < … < T_{S[N-1]}$, where "<" denotes the alphabetical order.

In Figure 3.10 we outline an algorithm which returns the position in string T (for which a suffix array S has been built) at which string X occurs. The search is performed in O(Plog$N$)) time, where P is the length of X and N is the length of T [Manber and Myers, 94]

[20] Although Zamir's algorithm (STC) is based on suffix trees, he points out that suffix arrays can be used for efficient construction of suffix trees.

Figure 3.10
Suffix array searching algorithm pseudo-code

```
/**
 * Suffix array searching algorithm
 *
 * T is the searched string
 * S is a Suffix Array for string T
 * X is the substring to be located
 */

if (X <= T[S[0]])
{
  xPosition = 0;
}
else if (X > T[S[N-1]])
{
  xPosition = N;
}
else
{
  L = 0;
  R = N - 1;

  while (R - L > 1)
  {
    M = (L + R) / 2;

    if (X <= T[S[M]])
    {
      R = M;
    }
    else
    {
      L = M;
    }
  }
  xPosition = R;
}

return xPosition as a result;
```

In [Manber and Myers, 94] a variation of the above algorithm is presented that uses additional information about the longest common prefixes (LCP) between suffixes to improve its time efficiency to $O(P+\log N)$ in the worst case.

3.3.2 Suffix array construction algorithms

One of the most important and interesting problems concerning suffix arrays is the space- and time-efficient construction of this data structure, which is very often referred to as the **suffix sorting** problem. In the simplest, naive, approach, the problem can be solved using a generic comparison-based sorting algorithm such as Quicksort. In this case, assuming the sorting algorithm performs $O(N\log N)$ comparisons, suffix sorting would require $O(N^2\log N)$ time[21]. The drawback of the generic sorting algorithm is that it does not exploit the specific characteristic of the texts being sorted: they all are suffixes of the same string. In this section we give a brief outline of two specialised suffix sorting algorithms that attain $O(N\log N)$ time complexity in the worst case.

[21] The sorting algorithm performs $O(N\log N)$ string comparisons, each comparison requiring $O(N)$ time in the worst case.

Manber-Myers algorithm

Manber and Myers propose [Manber and Myers, 94] an algorithm based on the bucket sort technique [Cormen et al., 90]:

> *The algorithm takes advantage of the fact that each suffix is a prefix of another one. Thus, the order of suffixes in the previous sorting pass is used as the keys for preceding suffixes in the next pass, each time doubling the number of considered symbols per suffix. This yields an algorithm which takes O(*NlogN*) time in the worst case.* [Larsson, 98]

In Figure 3.11 main steps of the algorithm are summarised:

Figure 3.11 *Manber-Myers suffix sorting algorithm pseudo-code [Larsson, 98]*

```
/**
 * Manber-Myers suffix sorting algorithm
 *
 * S is the suffix array to be sorted
 */

fill S with the numbers 0, ..., N-1;

bucket sort S using x_i as the key for i.
Regard S as partitioned into as many buckets
as there are distinct symbols in T;

k = 1;

while (all suffixes are not in separate buckets)
{
  for i = n-k, ..., n-1
  {
    move suffix S[i]+k to the beginning of its bucket
    and mark the position for bucket splitting;
  }

  for i = 0, ..., n-k-1
  {
    move suffix S[i]+k to the beginning of its bucket;
    when i crosses a bucket limit, mark accessed buckets for splitting;
  }

  split the marked buckets;

  k = k*2;
}
```

Sadakane algorithm

Sadakane noticed that with real-world data the Manber-Myers algorithm may behave inefficiently:

> *[…] most of the elements of S are often sorted in one of the first few passes of the algorithm, leaving only a few groups to be sorted by subsequent passes. Still, all elements of S are traversed during each pass.* [Larsson, 98]

Sadakane [Larsson and Sadakane, 99] improved the practical behaviour of the Manber-Myers algorithm by replacing the bucket sort technique with a comparison-based algorithm:

Figure 3.12
Sadakane suffix sorting algorithm pseudo-code [Larsson, 98]

```
/**
 * Manber-Myers suffix sorting algorithm
 *
 * S is the suffix array to be sorted
 */

sort S using x_i as the key for i.
Regard S as partitioned into as many buckets
as there are distinct symbols in T;

k = 1;

while (there are groups larger than 1)
{
  sort each group of size larger than 1 with a comparison-based
  algorithm, using the first position of the group containing
  x_{i+k} as the key for i when i+k < n, and -1 otherwise;

  split groups between non-equal keys;

  combine sequences of unit-size groups so that these can be skipped
  over in subsequent passes;

  k = k*2;
}
```

The trivial upper bound on the time complexity of the Sadakane algorithm is $O(N(\log N)^2)$. A more detailed analysis [Larsson, 98], however, reveals that the worst case complexity of $O(N \log N)$ is possible.

If space requirements are not crucial, a suffix-link based suffix tree construction algorithm [Ukkonen, 95] can be used to build a suffix array in $O(N)$ time. For a comprehensive review of other suffix sorting algorithms along with detailed benchmark results we refer the Reader to [Manzini and Ferragina, 02]. In [Baeza-Yates et al., 99] incremental and disk-optimised suffix array construction algorithms are presented.

4 LINGO - The Algorithm

In this chapter we provide the details on our proposed web search results clustering algorithm, hereafter referred to as LINGO[22]. We start with explaining the general concept of the algorithm and its connections to the Latent Semantic Indexing. Next, we give insight into all phases of LINGO, providing rationale behind the design decisions we had made. We also discuss several implementation and efficiency issues. We conclude this chapter with an analysis of the strong and weak points of our algorithm, setting the directions of future work.

4.1 Overview

When designing a web clustering algorithm, special attention must be paid to ensuring that both contents and description (labels) of the resulting groups are meaningful to the users. The majority of currently used text clustering algorithms follow a scheme where cluster content discovery is performed first, and then, based on the content, the labels are determined. Unfortunately, this may result in some groups' descriptions being meaningless to the users, which in turn, is very often caused by the nonsensical content of the clusters themselves. To avoid such problems LINGO adopts a radically different approach to finding and describing groups.

Find cluster descriptions first

The general idea behind LINGO is to *first* find meaningful descriptions of clusters, and then, based on the descriptions, determine their content. Similar technique seems to be used by Vivisimo [Vivisimo, 02a] – an interesting search results clustering engine, though the details of their algorithm have not been publicised:

> *We use a specially developed heuristic algorithm to group – or cluster – textual documents. This algorithm is based on an old artificial intelligence idea: a good cluster – or document grouping – is one, which possesses a good, readable description. So, rather than form clusters and then figure out how to describe them, we only form describable clusters in the first place.* [Vivisimo, 02b]

In this approach crucial is the careful selection of cluster labels – the algorithm must ensure that the labels both differ significantly from each other and at the same time cover most of

[22] Label INduction Grouping algOrithm; According to the Collins English Dictionary, lingo is *a range of words or a style of language which is used in a particular situation or by a particular group of people*. We believe that every successful web search results clustering algorithm should speak its users' lingoes, or at least accommodate to them.

the topics present in the input collection. As we show further in this chapter, it is possible to find such labels using the Vector Space Model along with the Latent Semantic Indexing technique.

Abstract concepts

As we have explained in the previous chapter, the Latent Semantic Indexing aims at representing the input collection by means of the *concepts* found in the documents rather than the literal terms that appear in them. To achieve this, the original term-document matrix is approximated by a limited number of orthogonal factors – the column vectors of the SVD's *U* matrix (see Section 3.2 for details). Intuitively, the factors can be perceived as a set of **abstract concepts** each of which conveys some common meaning present in a subset of the input collection [Deerwester et al., 90]. From the LINGO's point of view, the concepts seem to be perfect candidates for cluster labels. Unfortunately, with real-world data, their representation provided by the LSI is difficult to understand for humans – in order to obtain concise and informative labels we need to extract the verbal meaning of the abstract concepts.

To the best of our knowledge, little research has been done to infer the meaning of the orthogonal factors discovered by the LSI. To some extent this might be justified by the fact that such knowledge is unnecessary in text retrieval tasks, where the LSI was initially applied:

> *We make no attempt to interpret the underlying factors, nor to "rotate" them to some meaningful orientation. Our aim is not to describe the factors verbally but merely to be able to represent terms, documents and queries in a way that escapes the unreliability, ambiguity and redundancy of individual terms as descriptors.* [Deerwester et al., 90]

In [Kontostathis and Pottenger, 02] a detailed analysis is presented of how LSI finds and measures the relationships between single terms appearing in a collection of documents. The analysis is based on the notion of **term co-occurrence**:

> *Assume that a collection has one document that contains the terms "Java" and "applet". Also, assume that a different document in the collection contains the terms "Java" and "e-commerce". Furthermore, assume that "applet" and "e-commerce" do not co-occur elsewhere in the collection. It may be useful to gather these three terms in one unique class and infer that "e-commerce" and "applet" are actually related despite the fact that they do no co-occur in any item in the given collection. In other words, due to the strength of the statistical association of these terms, a limited degree of transitivity is imputed to the co-occurrence relation. This example of second-order co-occurrence an be extended to third, fourth or n^{th} order co-occurrence.* [Kontostathis and Pottenger, 02]

Kontostathis and Pottenger make a conclusion that term co-occurrences play a crucial role in the LSI and that they largely contribute to the high effectiveness of the LSI-based information retrieval systems.

Phrases to represent the abstract concepts

Among all term co-occurrences in a collection of documents **frequent phrases**[23] seem to be most meaningful to the users, being at the same time relatively inexpensive to find. As observed in [Weiss, 01] they are very often collocations[24] or proper names, which makes them both informative and concise. That is why we have decided to use them to represent the verbal meaning of the LSI's abstract concepts. Consequently, if at all possible, frequent phrases will be used in LINGO as cluster labels. Thus obtained descriptions will then become the base for determining the clusters' contents.

To assign documents to the already labelled groups LINGO could use the Latent Semantic Indexing in the setting for which it was originally designed: given a query – retrieve the best matching documents. When a cluster label is fed into the LSI as a query, as a result contents of the cluster will be returned. This approach should take advantage of the LSI's ability to capture high-order semantic dependencies in the input collection. In this way not only would documents that contain the cluster label be retrieved, but also the documents in which the same concept is expressed without using the exact phrase. In web search results clustering, however, the effect of semantic retrieval is sharply diminished by the small size of the input web snippets. This, in turn, severely affects the precision of cluster content assignment. That is why we have decided to use the simple Vector Space Model (see Section 3.1 for details) instead of the LSI to determine the cluster content.

To become a full-featured clustering algorithm, the process of finding cluster labels and contents must be preceded by some preprocessing of the input collection. This stage should encompass text filtering, document's language recognition, stemming and stop words identification. It is also recommended that post-processing of the resulting clusters be performed to eliminate groups with identical contents and to merge the overlapping ones. As a summary of the introductory discussion, in Figure 4.1 we present the main phases of LINGO.

Figure 4.1 *LINGO – main phases pseudo-code*

```
/** Phase 1: Preprocessing */
for each document
{
  do text filtering;
  identify the document's language;
  apply stemming;
  mark stop words;
}

/** Phase 2: Feature extraction */
discover frequent terms and phrases;

/** Phase 3: Cluster label induction */
use LSI to discover abstract concepts;
for each abstract concept
{
  find best-matching phrase;
}
prune similar cluster labels;
```

[23] Phrases that record at least a certain number of occurrences in the input collection.

[24] A collocation is a group of words for which usage has established such an affinity that fluent speakers automatically associate them together. Examples of English collocations can be such phrases as *rig the election* or *to cost an awful lot of money*.

```
/** Phase 4: Cluster content discovery */
for each cluster label
{
  use VSM to determine the cluster contents;
}

/** Phase 5: Final cluster formation */
calculate cluster scores;
apply cluster merging;
```

Noteworthy is the fact that all five phases of LINGO are independent and easily separable. This allows to manipulate the quality and resource requirements of the algorithm by providing alternative implementations of some of its components.

4.2 The algorithm

In this section we give details on our proposed implementation of all phases of LINGO. The implementation employs suffix arrays to discover frequent phrases and the Singular Value Decomposition (SVD) to obtain abstract concepts and cluster labels. The classic Vector Space Model is used to determine the content of the clusters.

4.2.1 Preprocessing

In web search results clustering, it is the web snippets that serve as the input data for the grouping algorithm. Due to the rather small size of the snippets and the fact that they are automatically generated summaries of the original documents, proper data preprocessing is of enormous importance. Although LSI is capable of dealing with noisy data, in a setting where only extremely small pieces of documents are available, this ability is severely limited. As a result, without sufficient preprocessing, the majority of abstract concepts discovered by the LSI would be related to meaningless terms, which would make them useless as cluster labels. Thus, the primary aim of the preprocessing phase is to remove from the input documents all characters and terms that can possibly affect the quality of group descriptions.

In LINGO there are four steps to the preprocessing phase: text filtering, document's language identification, stemming and stop words marking.

Text filtering

In the text filtering step, all terms that are useless or would introduce noise in cluster labels are removed from the input documents. Among such terms are:

- HTML tags (e.g. <table>) and entities (e.g. &)
- non-letter characters such as "$", "%" or "#" (except white spaces and sentence markers such as '.', '?' or '!')

Note that at this stage the stop-words are not removed from the input documents. Additionally, words that appear in snippet titles are marked in order to increase their weight in further phases of clustering.

Language identification

Before proceeding with stemming and stop words marking, for each input document separately, LINGO tries to recognise its language. In this way, for each snippet, appropriate stemming algorithm and stoplist can be selected. This step is immensely important for two main reasons. First of all, it may be inconvenient for the users to choose manually the appropriate language version of the clustering algorithm. With the automatic language recognition there is no need for the users to bother with such choice. Secondly, for many queries it is unreasonable to stick to one language only as documents in a mixture of different languages may be returned. In such case, language recognition enables proper identification of all stop words and makes it possible to prevent them from appearing in the cluster labels.

In LINGO we have decided to use the small word technique of language identification (see Section 2.2.5). Figure 4.2 illustrates the main idea of the technique:

Figure 4.2 *LINGO – language identification algorithm pseudo-code*

```
for each document
{
  for each available stoplist
  {
    count the occurrences of terms from the stoplist
    in the document;
  }

  choose the stoplist that recorded the highest number of occurrences;
  if (the highest number of occurrences > 1)
  {
    decide that the document's language is the
    stoplist's language;
  }
  else
  {
    decide that the document's language is unidentified;
  }
}

optionally, to the documents whose language is unidentified assign
the most popular language among the other documents;
```

Clearly, in order for the algorithm to identify a language, a stoplist for that language must be available. Alternatively, a trigram-based language recognition scheme can be used [Grefenstette, 95].

Stemming

In this step, if the required stemmer is available[25], inflection suffixes and prefixes are removed from each term appearing in the input collection. This guarantees that all inflected forms of a term are treated as one single term, which increases their descriptive power. At

[25] In the present implementation of LINGO only English and Polish stemming are supported. We use a free Java implementation of Porter stemmer [Porter, 80] for English and Lametyzator – a free stemming engine for the Polish language [Weiss, 03b].

the same time, as bare stems may be more difficult for the users to understand, for the presentation purposes the original (inflected) forms of the words are also stored.

Stop words marking

Although they alone do not present any descriptive value, stop words may help to understand or disambiguate the meaning of a phrase (compare: "Chamber Commerce" and "Chamber of Commerce"). That is why we have decided to retain them in the input documents, only adding appropriate markers. This will enable the fuhrer phases of the algorithm to e.g. filter out phrases ending with a stop word or prevent the LSI from indexing stop words at all.

4.2.2 Feature extraction

The aim of the feature extraction phase is to discover phrases and single terms that will potentially be capable of explaining the verbal meaning behind the LSI-found abstract concepts. To be considered as a candidate for a cluster label, a phrase or term must:

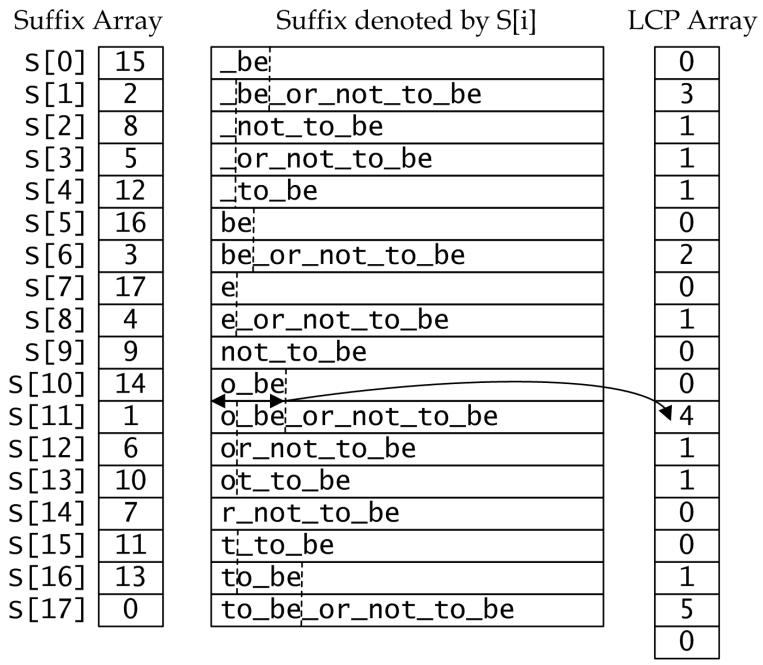
- Appear in the input documents at least a specified number of times. This is a result of a widely accepted assumption in information retrieval that features that recur frequently in the input documents have the strongest descriptive power. Additionally, omitting the infrequent words and phrases will significantly increase the time efficiency of the algorithm as a whole.
- Not cross sentence boundaries. Predominantly, sentence markers mark a topical shift, therefore a phrase extending beyond one sentence is likely to carry little meaning to the user [Zhang and Dong, 01].
- Be a complete phrase [Zhang and Dong, 01]. Compared to partial phrases, complete phrases should allow better description of clusters (compare "Senator Hillary" and "Senator Hillary Rodham Clinton"). For the definition of a complete phrase refer to Section 2.3.3.
- Not begin nor end with a stop word – stripping the phrases of leading and trailing common terms is likely to increase their readability as cluster labels. We again emphasise that stop words that appear in the middle of a phrase should not be discarded.

The phrase discovery algorithm

We have decided to use in LINGO a modified version of the phrase discovery algorithm proposed in [Zhang and Dong, 01]. The algorithm employs a variant of suffix arrays that is extended with an auxiliary data structure – an LCP[26] array. The array consists of N+1 integers while each element LCP[i] contains the length of the longest common prefix between adjacent suffixes S[i-1] and S[i] (see Figure 4.3). To simplify the algorithm, the LCP array is padded with zeros so that LCP[0] = LCP[N] = 0.

[26] Longest Common Prefix

Figure 4.3
An example extended suffix array



The phrase discovery algorithm works in two steps. In the first step, the right- and left-complete phrases (refer to Section 2.3.3 for details) are discovered, and in the second step, they are are combined into a set of complete phrases. In Figure 4.4 we provide an outline of the *discover_rcs* algorithm [Zhang and Dong, 01], which identifies all right-complete substrings of a string *T* in O(*N*) time[27], where *N* is the length of *T*.

Figure 4.4
The discover_rcs algorithm pseudo-code [Zhang and Dong, 01]

```
/**
 * The discover_rcs algorithm.
 *
 * lcp is the LCP array of T
 * N   is the length of T
 */
void discover_rcs(int [] lcp,
                  int N)
{
  /**
   * A structure that identifies a single
   * substring of T.
   */
  typedef struct
  {
    int position;
    int frequency;
  } SUBSTRING;

  /** A stack of right-complete substrings */
  SUBSTRING rcsStack[N];

  /** RCS stack pointer */
  int sp = -1;

  /** The position within the suffix array */
  int i = 1;

  while (i < N+1)
  {
    if (sp < 0) /** rcsStack is empty */
    {
```

[27] The algorithm assumes that a suffix array *S* for the string *T* has already been built.

```
      /**
       * If the current suffix has at least 2 occurrences
       * push it onto the stack.
       */
      if (lcp[i] > 0)
      {
        sp++;
        rcsStack[sp].position = i;
        rcsStack[sp].frequency = 2;
      }
      i++;
    }
    else
    {
      int r = rcsStack[sp].position;

      if (lcp[r] < lcp[i])
      {
        /**
         * Case A
         * The current prefix is longer. Push it onto the stack.
         */
        sp++;
        rcsStack[sp].position = i;
        rcsStack[sp].frequency = 2;
        i++;
      }
      else if (lcp[r] == lcp[i])
      {
        /**
         * Case B
         * The current prefix is of the same length.
         * Increase its frequency.
         */
        rcs_stack[sp].frequency++;
        i++;
      }
      else
      {
        /**
         * Case C
         * The current prefix is shorter. Output the prefix from the
         * top of the stack as a right-complete substring.
         */
        output rcsStack[sp];

        /**
         * Update the frequency of the longer prefixes starting with
         * the currently output prefix.
         */
        int f = rcsStack[sp].frequency;
        sp--;
        if (sp >= 0)
        {
          rcsStack[sp].frequency = rcsStack[sp].frequency + f - 1;
        }

        /**
         * If the stack is empty and the current prefix has at least 2
         * occurrences - put it onto the stack.
         *
         * This part was missing in the original algorithm, which
         * caused it to miscalculate the frequencies of some phrases.
         */
        if (lcp[i] > 0 && sp < 0)
        {
          sp++;
          rcsStack[sp].id = i;
          rcsStack[sp].frequency = 2 + f - 1;
          i++
        }
      }
    }
  }
}
```

The general idea behind the *discover_rcs* algorithm is to linearly scan the suffix array in search of frequent prefixes, counting their occurrences meanwhile. The central observation is that if LCP[*i*] > 0 for some *i* then the prefix of the suffixes *S*[*i*-1] and *S*[*i*] occurs at least twice in the string. Once such a prefix is identified, information about its position and frequency (initially the frequency is 2) is pushed onto the stack. When further scanning the suffix array, three specific cases, depicted in Figure 4.5, must be considered. In case A, the currently processed prefix is longer than the one on the top of the stack: |RCS*| < LCP[*i*], where |RCS*| denotes the length of the stack's top prefix. In such case the new longer prefix is put onto the stack, with initial frequency being equal 2. In case B, the current prefix is of the same length as the stack's topmost prefix, therefore the only action is to increase its frequency by 1. Case C, where the current prefix is shorter than the one on the top of the stack, is most complex. In this case, the stack's topmost prefix cannot be "continued" (case B) nor "extended" (case A) in any way, therefore it can be output as the resulting right-complete substring. Additionally, for the proper frequency calculation, the number of occurrences of the stack's next prefix must be increased accordingly (note that in case B only the frequency of the top prefix on the stack is increased). Finally, if the stack is empty after the removal of the output prefix, provided the current prefix is frequent, it is pushed onto the stack with the frequency corrected appropriately.

Figure 4.5
Three cases of the discover_rcs algorithm

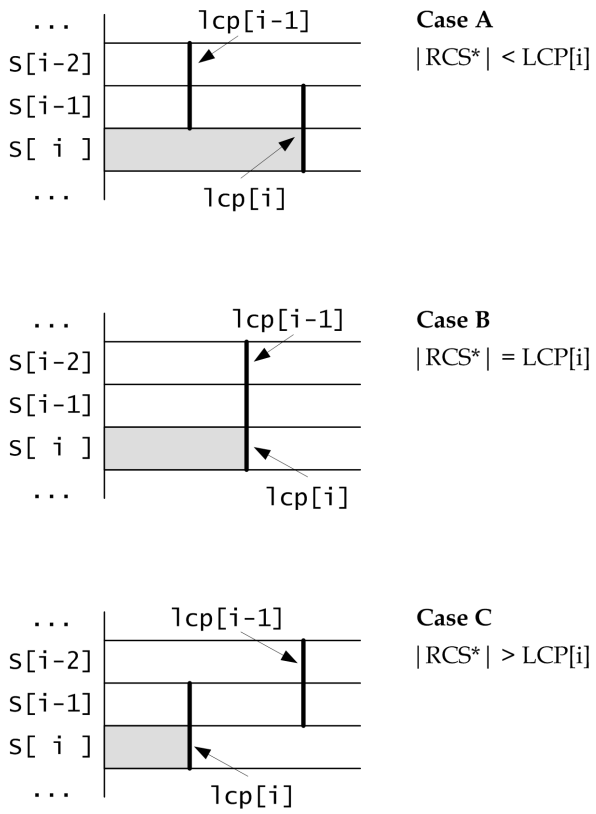


Figure 4.6 presents the output from the *discover_rcs* algorithm for the sample string shown in Figure 4.3.

Figure 4.6 *Example output of the discover_rcs algorithm*

Position	Frequency	Right-complete substring
1	2	_be
2	5	_
6	2	be
8	2	e
11	2	o_be
12	4	o
17	2	to_be
16	3	t

A complete substring must be both right- and left-complete. The left-complete substrings of *T* can be identified by applying the *discover_rcs* algorithm to ~*T* – the inverse of *T*. Then, using the *intersect_lcs_rcs* algorithm, the arrays of right- and left-complete substrings, RCS and LCS, can be combined to obtain the complete substrings. The algorithm requires that both input arrays be sorted alphabetically. While the RCS array is already ordered, the LCS array needs additional sorting. The *intersect_lcs_rcs* algorithm is outlined in Figure 4.7.

Figure 4.7 *The intersect_lcs_rcs algorithm pseudo-code [Zhang and Dong, 01]*

```
/**
 * The intersect_lcs_rcs algorithm.
 *
 * lcs is the array of left-complete substrings
 * rcs is the array of right-complete substrings
 * L   is the length of the lcs array
 * R   is the length of the rcs array
 */
void intersect_lcs_rcs(SUBSTRING [] lcs, SUBSTRING [] rcs,
                       int L, int R)
{
  int i = 0;
  int j = 0;

  while ( (i < L) && (j < R))
  {
    String lcSubstring = the substring denoted by lcs[i];
    String rcSubstring = the substring denoted by rcs[j];

    if (lcSubstring == rcSubstring)
    {
      output lcs[i];
      i++;
      j++;
    }

    if (lcSubstring < rcSubstring)
    {
      i++;
    }

    if (rcSubstring < lcSubstring)
    {
      j++;
    }
  }
}
```

In Figure 4.8 we present the output from the *intersect_lcs_rcs* algorithm for the sample string shown in Figure 4.3.

Figure 4.8 *Example output of the intersect_lcs_rcs algorithm*

Position	Frequency	Complete substring
2	5	_
12	4	o
16	3	t
17	2	to_be

In LINGO, the data on which the above algorithms work, is built of words, rather than single letters as in the examples. In this way, as a result, terms and phrases along with the number of their occurrences are returned.

In the final step of the feature extraction phrase, terms and phrases that exceed the **Term Frequency Threshold** are chosen. We have empirically established that for best accuracy of clustering, the value of the threshold should fall within the range between 2 and 5. The fairly low values of these boundaries result from relatively small sizes of the input documents (i.e. snippets). For a summary of all parameters of LINGO and their default values refer to Section 4.2.6.

Figure 4.9 summarises the whole feature extraction phase.

Figure 4.9 *LINGO – feature extraction phase pseudo-code*

```
/**
 * Phase 2: Feature extraction
 */

/** Conversion of the representation */
for each document
{
  convert the document from the character-based to
  the word-based representation;
}

/** Document concatenation */
concatenate all documents;
create an inverted version of the concatenated documents;

/** Complete phrase discovery */
discover right-complete phrases;
discover left-complete phrases;
sort the left-complete phrases alphabetically;

combine the left- and right-complete phrases into a
set of complete phrases;

/** Final selection */
for further processing choose the terms and phrases whose
frequency exceed the Term Frequency Threshold;
```

4.2.3 Cluster label induction

In the cluster label induction phase, meaningful group descriptions are formed based on the SVD decomposition of the term-document matrix. There are four steps to this phase: term-document matrix building, abstract concept discovery, phrase matching and label pruning and evaluation.

Term-document matrix building

In the initial step of the label induction phase, a term-document matrix A is built based on the input collection of snippets. Contrary to the SHOC approach[28], in LINGO only single terms are used to construct the A matrix. The reason for this is that it is more natural to use

[28] SHOC was designed with clustering of documents in Chinese in mind. In Chinese individual characters separated from their context carry no meaning, hence the need to use phrases (sequences of characters) and not single characters in the term-document matrix.

sets of single words rather than sets of phrases to express abstract concepts, which actually very often *are* phrases themselves. Additionally, individual terms allow finer granularity of description.

Terms that were marked as stop words or did not exceed the **Term Frequency Threshold** will not be included in the term-document matrix. The rest of the words will be reflected in the matrix using the **tf-idf scheme**. Additionally, as suggested in [Riboni, 02] weights of the terms appearing in snippet titles are scaled by a constant factor[29] s=2.5.

In Figure 4.10 an example collection of d=7 document titles is presented, in which t=5 terms and p=2 phrases appear more than once. The term-document matrix A is different from the matrix shown in Figure 3.2 as in this example the *tf-idf* weighting scheme was used. Indeed, for document D1, for example, the term "*Computation*" received a larger value than other terms, because it occurs in only one title, and hence the value of its *idf* factor is relatively high.

Figure 4.10 *Label induction example – input data*

The t=5 **terms**:
T1: Information
T2: Singular
T3: Value
T4: Computations
T5: Retrieval

The p=2 **phrases**:
P1: Singular Value
P2: Information Retrieval

The d=7 **documents**:
D1: Large Scale Singular Value Computations
D2: Software Library for the Sparse Singular Value Decomposition
D3: Introduction to Modern Information Retrieval
D4: Using Linear Algebra for Intelligent Information Retrieval
D5: Matrix Computations
D6: Singular Value Analysis of Cryptograms
D7: Automatic Information Organization

The 5x7 **term-document** matrix after column length normalization (tf-idf weighting):

$$A = \begin{pmatrix} 0.00 & 0.00 & 0.56 & 0.56 & 0.00 & 0.00 & 1.00 \\ 0.49 & 0.71 & 0.00 & 0.00 & 0.00 & 0.71 & 0.00 \\ 0.49 & 0.71 & 0.00 & 0.00 & 0.00 & 0.71 & 0.00 \\ 0.72 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.83 & 0.83 & 0.00 & 0.00 & 0.00 \end{pmatrix}$$

Abstract concept discovery

In this step the Singular Value Decomposition of the term-document matrix is performed in order to obtain the orthogonal basis of its column space. As discussed and shown earlier, particular vectors of the basis (SVD's U matrix) represent the abstract concepts appearing in the input documents. It should be noted, however, that not all column vectors of the U matrix will be considered in the process of label discovery. Similarly to the SVD-based document retrieval, only the first k basis vectors shall be used and it will be the U_k matrix that will be processed in the further phases.

To calculate the value of k, from the methods presented in Section 3.2.2 we have chosen the one based on the Frobenius norms of the term-document matrix and its k-rank approx-

[29] In [Riboni, 02] scaling factors ranging from 2 to 6 were examined.

imation. The primary motivation for our choice was the fact that the other approaches cannot be easily parametrised. A similar method was adopted in SHOC.

The selected method requires that a percentage quality threshold q be assumed that determines to what extent the k-rank approximation should retain the original information. In this way, k is set to the minimum value that satisfies the following condition:

$$\frac{\| A_k \|_F}{\| A \|_F} = \frac{\sqrt{\sum_{i=1}^{k} \left(\sigma_i^2\right)}}{\sqrt{\sum_{i=1}^{r_A} \left(\sigma_i^2\right)}} \geq q$$

In the above formula, A is the original term-document matrix, A_k is its k-rank approximation, r_A is the rank of the A matrix, σ_i is its ith singular value (the ith diagonal element of the SVD's Σ matrix) and $||A||_F$ denotes the Frobenius norm[30] of the A matrix.

Clearly, the larger the value of q the more cluster label candidates will be induced. The choice of the optimal value for this parameter ultimately depends on the users' preferences. That is why we have decided to make q one of the LINGO's parameters – **Candidate Label Threshold**. For a summary of all parameters of the algorithm and their default values refer to Section 4.2.6.

In figure Figure 4.11 the U and Σ matrices obtained from the SVD decomposition of the example term-document matrix (see Figure 4.10) are shown. Assuming 0.9 as the value for the q parameter, k=2 is the number of abstract concepts that shall be taken into consideration in the following steps.

Figure 4.11 *Label induction example – Singular Value Decomposition results*

$$U = \left(\begin{array}{cc|ccc} 0.00 & 0.75 & 0.00 & -0.66 & 0.00 \\ 0.65 & 0.00 & -0.28 & 0.00 & -0.71 \\ 0.65 & 0.00 & -0.28 & 0.00 & 0.71 \\ 0.39 & 0.00 & 0.92 & 0.00 & 0.00 \\ 0.00 & 0.66 & 0.00 & 0.75 & 0.00 \end{array}\right) \qquad \Sigma = \left(\begin{array}{cc|ccc} 1.64 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 1.56 & 0.00 & 0.00 & 0.00 \\ \hline 0.00 & 0.00 & 1.14 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.75 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{array}\right)$$

Phrase matching

This step relies on an important observation that both the abstract concepts and the phrases discovered in the feature extraction phase are expressed in the same space – the column space of the original term-document matrix. Thus, the classic cosine distance can be used to calculate how well a phrase or a single term represents an abstract concept.

To do so, let us define the $t \times (p+t)$ P matrix that will express the phrases and terms in the column space of the term-document matrix. Such matrix can be easily built treating the phrases and single words as pseudo-documents and using one of the term weighting schemes. In Figure 4.12 the column length-normalised form of the P matrix for the example collection of document titles and *tf-idf* term weighting is shown.

[30] For details regarding the norm refer to Appendix B

Figure 4.12
Label induction example – the P matrix

$$P = \begin{pmatrix} 0.00 & 0.56 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.71 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 \\ 0.71 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.83 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix}$$

Having the P matrix and the ith column vector of the SVD's U matrix, a vector m_i of cosines of the angles between the ith abstract concept vector and the phrase and term vectors can be calculated: $m_i = U_i^T P$. The phrase or term that corresponds to the maximum component of the m_i vector should be selected as the verbal representation of ith abstract concept. Additionally, the value of the cosine becomes the score of the cluster label candidate.

The phrase matching process for a single abstract concept can be extended to the whole U_k matrix. In this way a single matrix multiplication $M = U_k^T P$ yields a matrix of the cosines for all abstract concept – phrase pairs. Figure 4.13 presents the M matrix for the example document collection and the resulting cluster label candidates E1 and E2 along with their scores.

Figure 4.13
Label induction example – the M matrix

$$M = U_k^T P = \begin{pmatrix} \mathbf{0.92} & 0.00 & 0.00 & 0.65 & 0.65 & 0.39 & 0.00 \\ 0.00 & \mathbf{0.97} & 0.75 & 0.00 & 0.00 & 0.00 & 0.66 \end{pmatrix}$$

The e=2 **candidate cluster labels**:
E1: Singular Value (score: 0.92)
E2: Information Retrieval (score: 0.97)

As it may happen that neither of the frequent phrases will be able to describe an abstract concept better than a single term, it is important that not only phrases but also single words be considered at this stage. Noteworthy is the fact that due to the length-normalisation, a two-word phrase may record a lower similarity score than the single words of which it consists.

For the time being, to represent an abstract concept only the best-matching feature (word or phrase) is selected. Other features, some of which may be almost as close to the abstract concept as the winning one, are disregarded at this stage. We have decided to adopt such approach to ensure that each abstract concept is described by one and only one phrase or word. Additionally, such matching method greatly simplifies the algorithm. In Section 4.6, where we discuss the future improvements of LINGO, we show an alternative to this simple strategy.

Candidate label pruning

Although the SVD-derived basis of the column space of the term-document matrix is orthogonal, some of the cluster label candidates (verbal representations of the abstract concepts) tend to overlap. The reason for this is that for each abstract concept only the best-matching word or phrase is selected and other phrases, which potentially could help express the difference between two abstract concepts, are ignored.

It is therefore necessary to discard those of the candidate cluster labels that are too similar. To calculate the similarity we will again employ the classic Vector Space Model – the cosine of the angle between two cluster label vectors (i.e. the column vectors of the P matrix) will characterise the strength of the relationship between them. From a group of labels whose cosine similarity is higher than **Label Similarity Threshold**, only the one with the highest score will be retained. We have empirically established that the value of this threshold should range from 0.2 to 0.5. For a summary of all parameters of LINGO and their default values refer to Section 4.2.6.

As a summary of the cluster label induction phase, in Figure 4.14 we present it in the form of pseudo-code.

Figure 4.14
LINGO – cluster label induction phase pseudo-code

```
/**
 * Phase 3: Cluster label induction
 */

/** Term-document matrix building */
build the term-documen matrix A for the input snippet collection.
as index terms use the non-stop words that exceed the predefined
term frequency threshold. use the tf-idf weighting scheme;

/** Abstract concept discovery */
perform the Singular Value Decomposition of the term-document
matrix to obtain U, S and V matrices;

based on the value of the q parameter and using the S matrix -
calculate the desired number k of abstract concepts;

use the first k columns of the U matrix to form the Uk matrix;

/** Phrase matching */
using the tf-idf term weighting create the phrase matrix P;
for each column of the Uk matrix
{
  multiplicate the column by the P matrix;
  find the largest value in the resulting vector to determine
  the best matching phrase;
}

/** Candidate label pruning */
calculate similarities between all pairs of candidate labels;
form groups of labels that exceed a predefined similarity threshold;
for each group of similar labels
{
  select one label with the highest score;
}
```

4.2.4 Cluster content discovery

In the cluster content discovery phase, the classic Vector Space Model is used to assign the input snippets to the cluster labels induced in the previous phase. The assignment process much resembles document retrieval based on the VSM model (see Section 3.1.2) – the only difference is that instead of one query, the input snippets are matched against every single cluster label. Thus, in place of the VSM's query vector q, let us define the matrix Q, in which each cluster label is represented as a column vector in the same way as it was in the phrase matrix P (see previous section). Further, let $C=Q^TA$, where A is the original term-document

matrix. In this way, element c_{ij} of the *C* matrix indicates the strength of membership of the *j*th document in the *i*th group.

A snippet will be added to a cluster if the corresponding element of the *C* matrix exceeds the **Snippet Assignment Threshold**. From our observations, values of the threshold falling between 0.15 and 0.30 yield clusters of best quality (for details on evaluation of LINGO refer to Chapter 5). For a summary of all parameters of our algorithm and their default values refer to Section 4.2.6.

In a natural way, the assignment scheme based on cosine similarity has the ability to create overlapping clusters. Additionally, values of the *C* matrix, can be used to sort snippets within their groups, so that the most relevant ones will be easier to identify. Finally, as it is possible that some snippets may match neither of the cluster labels, a special group labelled e.g. "Other topics" can be created in which such snippets should be placed.

In Figure 4.15 we present matrix *C* and the resulting clusters for the example collection of document titles.

Figure 4.15
Cluster content discovery example

$$Q = \begin{pmatrix} 0.00 & 0.56 \\ 0.71 & 0.00 \\ 0.71 & 0.00 \\ 0.00 & 0.00 \\ 0.00 & 0.83 \end{pmatrix} \qquad C = Q^T A = \begin{pmatrix} 0.69 & 1.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 1.00 & 1.00 & 0.00 & 0.00 & 0.56 \end{pmatrix}$$

Information Retrieval [1.0]
Introduction to Modern Information Retrieval [1.00]
Using Linear Algebra for Intelligent Information Retrieval [1.00]
Automatic Information Organization [0.56]

Singular Value [0.95]
Software Library for the Sparse Singular Value Decomposition [1.00]
Singular Value Analysis of Cryptograms [1.00]
Large Scale Singular Value Computations [0.69]

Other Topics
Matrix Computations

4.2.5 Final cluster formation

In the final phase of the algorithm, cluster scores are calculated according to the following formula:

cluster-score = *label-score* * *member-count*

If in the presentation interface the resulting clusters are sorted according to their score, the user will presented with the well-described and relatively large groups in the first place. In Figure 4.15 cluster scores are normalised so that the highest score is 1.0.

For the time being, no cluster merging strategy is implemented in LINGO. Although for the majority of queries our algorithm performs well without such step (see Chapter 5), we

consider including the post-processing of the resulting groups in its future versions (see Section 4.6).

4.2.6 Algorithm parameters

A number of parameters determine the way LINGO identifies group labels and content. Table 4.1 summarises the parameters and their influence on the clustering results. The proposed values of each parameter have been determined empirically.

Table 4.1 Parameters of LINGO

Parameter	Proposed value	Description	Effect
Candidate Label Threshold	0.70 – 0.90 (default: 0.775)	Determines the maximum number of candidate cluster labels. Corresponds the the *k*-rank approximation quality threshold *q* – refer to Section 4.2.3	The closer the value of the parameter to 1.0 the more clusters will be created.
Snippet Assignment Threshold	0.15 – 0.30 (default: 0.15)	Determines the similarity threshold that must be exceeded in order for a document to be added to a cluster. Refer to Section 4.2.4 for the details.	The larger the value of this parameter, the less documents will be placed in each cluster, the bigger the "Other Topics" group and the higher the assignment precision. The lower the value, the more documents each cluster will contain, the smaller the "Other Topics" group and the higher assignment recall.
Term Frequency Threshold	2 – 5 (default: 2)	Determines how many times a term must appear in the input documents in order to be included in the term-document matrix.	The smaller the value of this parameter, the more accurate cluster labels will be induced and the longer will the calculations take (due to the larger size of the term-document matrix). The higher the value of this parameter, the worst quality of cluster labels and the faster processing.
Label Similarity Threshold	0.20 – 0.50 (default: 0.30)	Determines the acceptable level of similarity (cosine distance) between cluster labels. If the similarity exceeds this threshold, the label with lower score will be discarded.	The higher the value of this parameter, the more diverse labels will be generated. Setting Label Similarity Threshold to 0.0 is equivalent to disabling the cluster label pruning step (see Section 4.2.3.4 for details).

What is noteworthy is that of the above thresholds only the first two may need precise tuning in order to adjust the clustering results to the users' preferences. The other two play secondary role and will rarely have values different from the default ones.

4.3 LINGO and SHOC

As we have already emphasised, the direct inspiration for our algorithm came from Semantic Hierarchical On-line Clustering – a fairly new approach to web search results clustering proposed in [Zhang and Dong, 01]. Initially, we planned a mere re-implementation of SHOC, but the drawbacks to this algorithm we identified gave us the incentive to further work. As a result LINGO – an algorithm that very much differs from its predecessor – has been designed and implemented.

The most important similarity between LINGO and SHOC is the application of Singular Value Decomposition in the process of clustering. However, the SVD is used for different purposes in the two algorithms. In LINGO it facilitates the cluster *label* discovery phase, while in SHOC the SVD is used directly to discover cluster *content*. What should also be emphasised is that it is our algorithm that introduced the first-describe-then-cluster paradigm. In Table 4.2 we discuss the differences and similarities between LINGO and SHOC in detail.

Table 4.2 *Comparison of LINGO and SHOC*

	SHOC	**LINGO**
Preprocessing	Does not include language identification step	Includes snippet language identification step
Phrase identification	Based on Suffix Arrays	Adapted from SHOC
Label discovery	Performed *after* cluster content discovery	Based on Singular Value Decomposition, performed *before* cluster content discovery
Cluster content discovery	Based on Singular Value Decomposition	Based on cluster labels, employs the Vector Space Model
Post-processing	Hierarchical cluster merging based on content overlap	No cluster merging applied

4.4 The implementation

As part of this thesis we prepared a practical implementation of LINGO. In this section we briefly outline the framework within which the implementation operates and make a few remarks on selected programming aspects of LINGO.

4.4.1 Carrot² Framework

The implementation of LINGO is based on the Carrot² framework developed by Dawid Weiss [Weiss, 03]. Carrot² is an open-source environment that facilitates experiments with processing and visualisation of web search results. Carrot's architecture is based on a set of distributed components that cooperate with each other by exchanging XML data.

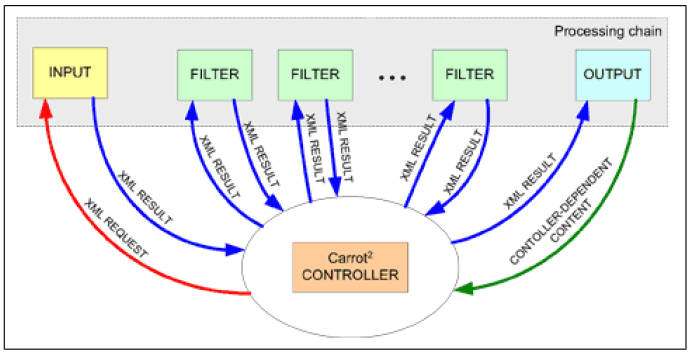


Figure 4.16 Carrot² components data flow

Three general types of components are available: input, filter and output components. The task of an input component is to deliver data for further processing. An example input component could, for example, fetch the results generated by some web search engine in response to the Carrot's user query. Filter components transform their input data in some way. An example transformation could be stemming or clustering of the input data. Clearly, in Carrot², LINGO acts as a filter component. Finally, the role of an output component is to present its input data to the user. Below we present a Carrot² setting in which web search results clustering can be performed. In Figure 4.18 an example of a tree-based presentation of clustering results is shown.

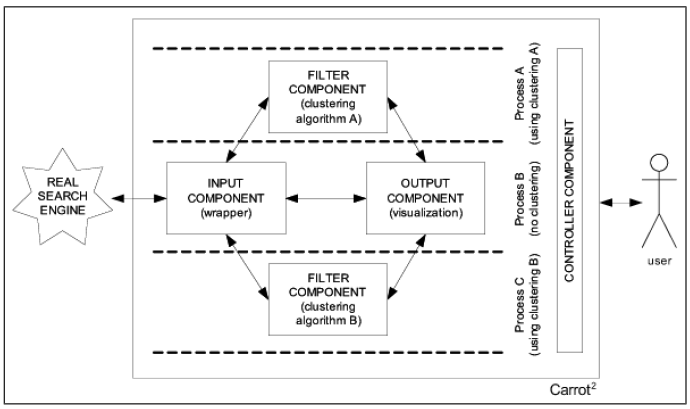


Figure 4.17 Carrot² clustering process

The current version of Carrot^2 and LINGO should be available at the following address:

http://ophelia.cs.put.poznan.pl:2001/

Figure 4.18 *Carrot^2 clustering results presentation*

4.4.2 Efficiency

For many reasons we have decided to implement LINGO in Java. First of all, the language was used to build Carrot^2, and hence LINGO could reuse some of its software components[31]. Secondly, with its emphasis on Object Oriented Programming (OOP), Java enforces the much needed *good software practices* such as the use of interfaces and proper code organisation. Finally, Java comes with a package of optimised ready-to-use data structures (such as hashtables or lists) that make programming faster and less error-prone.

The speed of Java

Java has been very often criticised for the slow speeds of execution, incomparable with those of a natively-compiled code. While this is still objectively true, with the recent optimisations, such as the Java HotSpot[32], Java code can achieve the running times comparable with those of e.g. C++. As shown in [Stern, 02] this is especially the case when the the Java built-in datatypes are used[33]. For this reason we have decided to design the implementation

---

[31] Technically, as the Carrot^2 components communicate through the HTTP protocol, it is possible to implement them in other languages (such as e.g. C++). In this case, however, much more coding would be needed to provide network communication, XML parsing etc.

[32] The HotSpot technology encompasses such techniques as the Just In Time (JIT) compilation of the byte-code, so that the most frequently used fragments of the program are executed much faster. For the details regarding Java HotSpot refer to [HotSpot, 03].

[33] In sorting an array of integers the C++ version of the algorithm turned out to be less than 30% faster than the Java's Arrays.sort() method.

of LINGO in the way that the most time-consuming operations are performed on such types.

It is for the performance reasons that we have decided to represent the input documents as integer values. In this way, every distinct stem of a word is assigned a unique integer identifier. Likewise, sentence markers (such as "." or "?") are also represented by integer values, but this time the codes are different for each occurrence. This assures that the discovered phases do not cross sentence boundaries. In Figure 4.19 an example of the integer-code representation is shown.

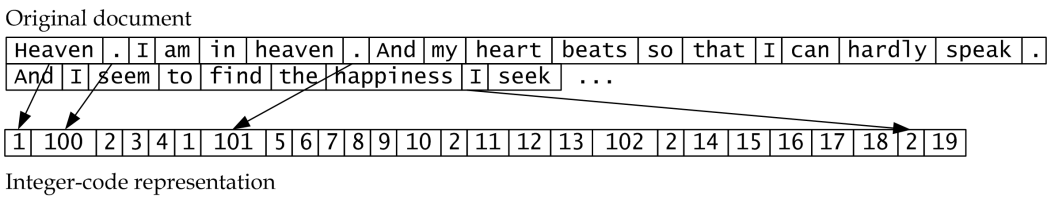


Figure 4.19 *Integer-code representation of the input documents*

What should be emphasised about the integer-code representation is that it allows much faster suffix sorting. The main reason is that the algorithm's input "string" is much shorter, as each word is represented by a single symbol[34]. Additionally, in the integer-code representation two words can be compared using a single integer comparison operation[35].

In the cluster label induction phase a significant amount of matrix computation (mostly transpositions and multiplications) is performed. We have decided to use an external library – JAMA[36] – to avoid the independent implementation of the Singular Value Decomposition.

As a summary of our algorithm's efficiency discussion, in Table 4.3 we present processing times of all of its phases. The figures have been averaged over ten queries, each of which returned approximately 100 snippets. The measurements were done on an Athlon 1.3GHz/768MB machine.

Table 4.3 *LINGO running times*

Phase	Average Time [s]	Fraction of the total
Preprocessing	0,071	9%
Feature Extraction	0,046	6%
Singular Value Decomposition	0,572	73%
Label Induction and Cluster Assignment	0,092	12%
TOTAL	**0,781**	

[34] Clearly, the reduction in the length of the input stream is done at the cost of the increased size of the alphabet. The majority of suffix sorting algorithms however, work well regardless of the alphabet size.

[35] Noteworthy is the fact that the suffix sorting algorithm does not require that the element codes be sorted alphabetically. The only requirement is that some *consistent* order is present, e.g. the order of appearance in the input text.

[36] Java Matrix Package, refer to [JAMA, 03] for the details.

As it might have been expected, it is the Singular Value Decomposition that accounts for over 70% of the total computational expense. Noticeable is also the insignificant contribution of the feature discovery phase towards the overall clustering time.

4.5 Algorithm discussion

In this section we discuss the strong and weak points of LINGO and their possible effect on the quality of clustering. We feel that the following advantages of our algorithm should be emphasised:

a. **Readable cluster labels**

The use of phrases in the process of cluster label induction guarantees that group descriptions can be easily understood by the users. As argued in [Zamir, 99], frequent phrases significantly increase the overall quality of clustering, not only of the phrase-based algorithms (such as Suffix Tree Clustering) but also of other approaches such as k-means. Similar effects can be observed also in LINGO.

b. **Diverse cluster labels**

Apart from the general abstract concepts related to fairly large groups of documents, Latent Semantic Indexing discovers narrower, more-specific ones. In this way meaningful clusters can be created whose labels are not necessarily the highest-frequency phrases. Additionally, the orthogonality of the SVD-derived abstract concept vectors, makes the diversity among cluster labels even wider.

c. **Overlapping clusters**

Placing the same document in a larger number of clusters increases the chances that, viewing only selected groups, the user will be able to identify all relevant documents. Moreover, some snippets may be related to two or more topics and thus should be placed in all respective clusters.

d. **Language independence**

Language recognition step helps LINGO prevent meaningless phrases or terms, such as stopwords, from appearing in cluster labels. Additionally, a single version of the clustering algorithm can be used for all sorts of queries, which may turn out to be much more convenient for the users.

e. **Ease of tuning**

Only two major parameters determine the way LINGO clusters the input collection. One of them enables to adjust the number of groups, the other – the cluster assignment precision. In this way, the algorithm's behaviour can be easily tailored to the particular users' preferences.

f. **Modular design**

As noted previously, all phases of LINGO are easily separable. Thus, it is possible to provide alternative implementations of some of them, improving the quality or time-efficiency of the algorithm as a whole.

As shown in the next chapter, some of the above features significantly contribute to LINGO's efficiency in web search results clustering tasks. Nonetheless, we are aware of a number of serious limitations that LINGO has in its present form:

a. **Flat clusters**

For the time being, LINGO is unable to deliver a hierarchical structure of thematic groups, which may result in some clusters being too general and too large.

b. **"Fixed" number of clusters**

The number of resulting clusters only to a very limited extent depends on the *actual* number of topics present in the input collection. In this way, it is possible that for some queries, the fraction of unintuitive or even meaningless clusters will be significant.

c. **"Other topics" problem**

In the present implementation of LINGO, regardless of the query, the number of induced group labels depends on the value of the Candidate Cluster Threshold, which makes the value fixed to a large extent. Thus, for a collection that contains much more topics than the precalculated number of candidate labels, many of the documents may be classified as "Other topic" even though a separate group should have been created for them.

d. **Overspecialised cluster labels**

The use of the classic Vector Space Model in the cluster content discovery phase may cause some clusters to contain irrelevant documents, i.e. such documents that match only a part of the phase used in the description.

e. **Computational requirements**

Being based on the Singular Value Decomposition, LINGO is a computationally expensive algorithm. A set of documents larger than 150–200 items may take longer than 3 seconds to cluster, even on a relatively fast machine. In a high-load environment such processing times may be unacceptable.

f. **Lack of incremental processing**

In its present implementation, LINGO does not support incremental processing of the input snippets. For this reason its computational complexity affects even more severely the total time of query processing.

In the next section, where we discuss the future improvements of LINGO, we show possible remedies for the above deficiencies. For an analysis of real-world queries and human-made evaluation of LINGO's clustering results refer to Chapter 5.

4.6 Future improvements

While practical experiments (see Chapter 5) suggest that LINGO performs fairly efficiently in web search results clustering tasks, we feel that several extensions could be introduced to ensure even better quality of grouping. Below we give an overview of the proposed improvements.

a. **Hierarchical clustering**

Owing to its flexible design, there are several ways in which LINGO could be extended to support hierarchical groupings. In [Zhang and Dong, 01] a simple method based on clusters' content overlap is proposed (see Section 2.3.3 for details) which could be added as an additional step to LINGO's post-processing phase. Another approach could utilise the fact that, apart from document-query similarities, the Latent Semantic Indexing can calculate a matrix of term-term (or phrase-phrase) associations [Berry et al., 99]. Based on such a matrix, not only could the hierarchical structure rely on the actual content of groups, but also on the semantic relationships between their labels.

b. **Incremental processing**

In order to achieve a noticeable improvement in LINGO's response time, the incremental model of processing would have to be introduced primarily at the Singular Value Decomposition stage. This, however, requires that all preceding phases must also perform their computations in a similar manner.

The preprocessing phase is by its very nature a sequential process, and hence it should not present any difficulties while designing its incremental version.

The feature extraction phase however, is more difficult to modify. In the present implementation of LINGO, phrase discovery is based on suffix arrays, therefore an incremental method of building the data structure would be needed. In [Baeza-Yates et al., 99] suffix array construction techniques for large texts are discussed in which the input text is split into smaller blocks. In this way, suffix sorting can be performed on each block separately, but at the same time the partial results can be successively merged into the final suffix array.

An incremental implementation of the cluster label induction phase requires that the Latent Semantic Indexing be performed in the same way. In [Berry et al., 95] two techniques – Folding-In and SVD-Updating – are presented that allow adding new documents to the already built k-rank approximation of the term-document matrix without recomputing the SVD from scratch. While the accuracy

of these methods is slightly worse than the explicit SVD recomputation, they are much more time-efficient.

As it is the Singular Value Decomposition that accounts for more than 70% of LINGO's running time (see Table 4.3), an alternative probabilistic SVD algorithm [Drineas et al., 99] could at all eliminate the need for incremental label induction. Another approach could employ Concept Indexing [Karypis and Han, 00] – an effective dimensionality reduction method based on the k-means algorithm.

c. **Cluster label induction**

There are several ways in which the quality of the induced group labels could be improved. The problem that should be addressed in the first place is the proper (i.e. more dynamic) calculation of the optimal number of clusters. Unfortunately, for the time being, we have no clear idea that could be put under discussion.

Secondly, alternative methods of extracting the verbal meaning of the LSI-derived abstract concepts should be considered. One of such methods may be allowing more than one phrase to appear in a single group description. Another approach would be that instead of pruning of similar labels, the second or third matching feature is taken as the verbal representation of an abstract concept.

Thirdly, variations of the basic Latent Semantic Indexing can allow better identification of the abstract concepts themselves. In [Ando, 00] a method based on singular value computation is proposed that, according to the author's experiments, yields a better basis of the input collection's column space. Finally, Natural Language Processing methods could be employed to prune those of the candidate labels that are e.g. unlikely to form a grammatically correct phrase.

d. **Cluster content assignment**

The concept of semantic clustering, in which group members are chosen based not only on simple lexical matching, is still not fully exploited by LINGO, and thus very intriguing. Therefore, the application of Latent Semantic Indexing to discovering the actual cluster content should be reexamined. One of possible ways in which the problem of low precision of such assignment could be circumvented is replacing the fixed value of the Snippet Assignment Threshold with an adaptative algorithm.

5 LINGO - Evaluation

In this chapter we present our attempts to evaluate the web search results clustering algorithm we had created. We have decided to assess LINGO in two opposing aspects. Firstly, for a number of real-world queries, we subjectively analyse the results obtained from our algorithm with respect to certain characteristics of the input data. Secondly, we propose a more formal user-based scheme for the evaluation of text clustering results. In this way, comparisons between different algorithms will be possible. Finally, we report on LINGO's performance according to our proposed assessment method.

5.1 Evaluation of search results clustering

As we have constantly emphasised throughout this thesis, it is with users in mind that web search results clustering algorithms should be created. In such an approach the *usefulness* of the generated document groups determines the overall algorithm's performance. Unfortunately, the term *usefulness* involves very subjective judgements of the clustering results. The same set of groupings might be perceived as "good" by some users, while at the same time being less attractive to others. This discrepancy can be partially explained by the different preferences the users might have as to the "ideal" results – some of them will prefer only a few larger clusters whereas other users may like a greater number of smaller groups. Therefore, the problem of automatic evaluation of the real-world performance of text clustering algorithms is particularly difficult [Stefanowski and Weiss, 03], if at all possible[37], to solve. Still, however crude the already existing automatic methods may be, we need them to draw statistically justified conclusions, to compare different clustering approaches or to perform large scale experiments with e.g. algorithms' parameters.

Before we proceed with outlining the most common techniques of clustering evaluation, let us define the two popular Information Retrieval metrics: **precision** and **recall**. First, let us assume that from a set D of documents, in response to the user's query the set A of documents was returned. Further, let R denote the set of all documents in D that are relevant to the query. Finally, let R_A be the intersection of R and A.

> **Definition — Precision** is the fraction of the retrieved documents which is relevant: $precision = |R_A| / |A|$.

[37] Intuitively, we feel that if an effective automatic assessment of the usefulness of clustering results could be implemented, there should be no difficulty in creating a grouping algorithm that would meet all required criteria.

Definition — Recall is the fraction of the relevant documents which has been retrieved: $recall = |R_A| / |R|$.

Ideally, both of the measures would be equal to 1.0. In practice, however, a trade-off between them must be sought – in most situations an algorithm will attain higher precision at the cost of worse recall and vice versa. In [Baeza-Yates et al., 99] alternative measures of Information Retrieval quality are discussed.

Below we describe briefly the three most popular methods of clustering results evaluation.

a. **Standard IR metrics**

In this approach the quality of clustering is evaluated in terms of the standard precision and recall measures. One variant of such a method is where the contents of the top-scoring cluster is assumed to be the set of retrieved documents [Zamir, 99]. Provided that the set of relevant documents is known a priori or had been identified in advance, the precision and recall can be easily calculated.

In [Weiss, 01] the drawbacks of such approach have been pointed out. Among them are the lack of standard collections of web snippets and sometimes ambiguous human relevance judgements.

b. **Merge-then-cluster approach**

A common evaluation method is to test clustering algorithms on a specially prepared collection of texts [Zamir, 99]. Such a collection is usually created by merging smaller document groups each of which shares the same topic. An effective clustering algorithm should be able to identify the original clusters.

The reliability of this approach largely depends on the characteristic of the input collection. If the source groups share common topics, the evaluated algorithm may be unable to separate them, decreasing the overall value of the analysis. What should be emphasised, however, is that this type of evaluation can be performed on-demand and automatically.

c. **User evaluation**

This method of evaluation relies on the real users' opinions about the quality of clustering. The necessary data can be collected either from specially prepared questionnaires or from the web server logs. While this approach is potentially capable of verifying whether the clustering algorithm fulfils its users' needs, it still has some drawbacks:

> *Firstly, the level of users' experience greatly influences the time and accuracy of results they achieve (and hence system's evaluation of course). In case of a controlled study, the number of participants should be statistically significant. […] Users should also be involved in the process of evaluation and not simply "click-and-go", like it sometimes is the case when the process is carried out among students. Addition-*

ally, to perform quality evaluation, they should have at least basic knowledge of how to use the system [Weiss, 01].

One more obvious deficiency of user evaluation is that it cannot be performed automatically and on-demand. This makes that scheme unsuitable for experiments with e.g. algorithm's parameters.

5.2 User evaluation of LINGO

Despite its drawbacks, we have decided to use the user-based evaluation method to assess the clustering results produced by LINGO. Designing our algorithm we placed much emphasis on making the cluster labels accurate and meaningful to the *end users*. Therefore, we feel that is the users who can judge best to what extent LINGO achieved its goal. Below, we present the details on our proposed assessment scheme and report on the evaluation results.

5.2.1 The methodology

In our proposed assessment method we adapt the standard Information Retrieval metrics – precision and recall – to assess both the individual groups and the set of clusters as a whole. For a set of groups created in response to a single query, we ask the users to evaluate the following:

a. **Usefulness of clusters**

For each created cluster, based on its label, the user is asked to decide whether the cluster is useful or not. Useful groups would most likely have concise and meaningful labels, while the useless ones would have been given either ambiguous (e.g. too long) or senseless (e.g. consisting of a single stopword) descriptions.

Scale: useful group | useless group

b. **Assignment precision**

For each cluster individually, for each snippet from this cluster, the user is asked to judge the extent to which the snippet fits its group's description. A very well matching snippet would contain exactly the information suggested by the cluster label. A moderately matching snippet, despite being e.g. more general, would still be somehow related to the group's topic. A non-matching snippet, even though it might contain several words from the group's label, would be completely irrelevant to its cluster.

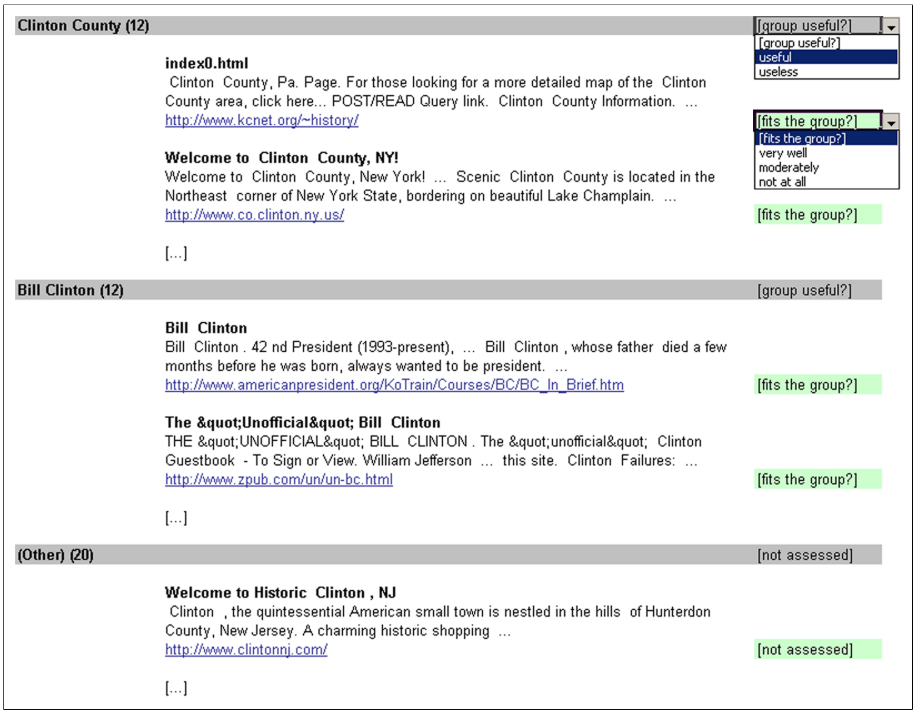
It is important that groups which have previously been assessed as useless be excluded from this step – if a user cannot understand the description of a group they will most likely be unable to evaluate its contents either. Similarly, clusters

containing unclassified documents (i.e. LINGO's "Other topics" group) should be disregarded at this stage.

Scale: very well matching | moderately matching | not matching

Below, we present a fragment of LINGO evaluation questionnaire for an example query.

Figure 5.1
A fragment of the evaluation form



Based on user feedback regarding a single set of clusters the following totals can be calculated:

g – the total number of created groups

u – the total number of groups judged as useful

a – the total number of snippet assignments (including groups judged as useless and the "Other topics" group). In case of overlapping clusters, due to the multiple assignments, a may be greater than the total number of input snippets.

w – the total number of snippet assignments judged as very well matching

m – the total number of snippet assignments judged as moderately matching

n – the total number of snippet assignments judged as non-matching

s – the total number of snippets in the processed search results set

o – the total number of snippets confined to the "Other topics" group

Using the above totals we define three measures characterising the quality of clustering:

a. **Cluster label quality**

Cluster label quality is the fraction of all created groups that the user judged as useful:

$$q = u/g$$

The values of q can range from 0.0 (no useful groups created) to 1.0 (all discovered groups are useful).

b. **Snippet assignment distribution**

Snippet assignment distribution consists of three components: the fraction of very well (p_w), moderately (p_m) and non-matching (p_n) snippets in relation to the total number of assessed assignments (i.e. the assignments to the useful groups):

$$p_w = w/(w + m + n)$$
$$p_m = m/(w + m + n)$$
$$p_n = n/(w + m + n)$$

The values of p_w, p_m and p_n range from 0.0 to 1.0, with an additional constraint: $p_w + p_m + p_n = 1.0$.

c. **Assignment coverage**

Assignment coverage is the fraction of snippets assigned to groups other than "Other topics" (i.e. including groups judged as useless) in relation to the total number of input snippets.

$$c = (s - o)/s$$

The value of c can vary from 0.0 (no assessable snippet assignments have been made) to 1.0 (all snippets have been assigned to clusters).

Additionally, we have decided to introduce one more parameter – the cluster overlap. Although it does not directly describe the quality of clustering, it may prove helpful in the analysis of the evaluation results.

d. **Cluster overlap**

Cluster overlap describes the fraction of snippets confined to more than one group:

$$v = a/s - 1$$

The values of v may vary from 0.0 (no overlapping assignments) up to infinity. However, as it is shown in the next section, in practice the values of v are unlikely to exceed 0.5 (which means that 50% of the input snippets have been assigned to more than one group).

Similarly to the tradeoff between precision and recall in the standard Information Retrieval evaluation, in our scheme the fraction of well-matched snippets *w* and assignment coverage *c* are a pair of conflicting measures. In case of LINGO, lowering the value of the Snippet Assignment Threshold (see Section 4.2.6) would increase the assignment coverage at the cost of lower number of well-assigned snippets (clusters would contain more documents, many of which might turn out irrelevant). Increasing the value of LINGO's Snippet Assignment Threshold would improve the snippets assignment lowering the coverage.

A clear advantage of the above evaluation scheme is that it can express ambiguous relevance judgements by means of standardised numerical values. Therefore, it can be be used both to measure the performance of individual clustering algorithms as well as to compare different methods. On the other hand, the major drawback to our scheme is that being based on user evaluation it is time-consuming and cannot be performed automatically or on-demand.

5.2.2 The results

In this section we report on the results of user evaluation of LINGO. Using our proposed assessment scheme, users evaluated clusters created in response to four queries, of which two were in English and two in Polish. We also tried to vary the spectrum of topics present in the results – two of the queries are rather general while the other two are more specific.

Query	Query language	Query type	Source search engine	Number of input snippets
clinton	English	general	Google	100
hall of residence in sheffield	English	specific	Google	84
stanisław lem	Polish	general	GooglePL	100
tarsjusz	Polish	specific	GooglePL	80

The values of LINGO's parameters remained constant for all queries and were the following:

Parameter	Value
Language identification	Yes
Candidate Label Threshold	0.775
Snippet Assignment Threshold	0.150
Term Frequency Threshold	2
Label Similarity Threshold	0.30

Due to the limited amount of time allocated to this project we managed to collect only seven evaluation forms[38], hence the assessment results may not be statistically representative.

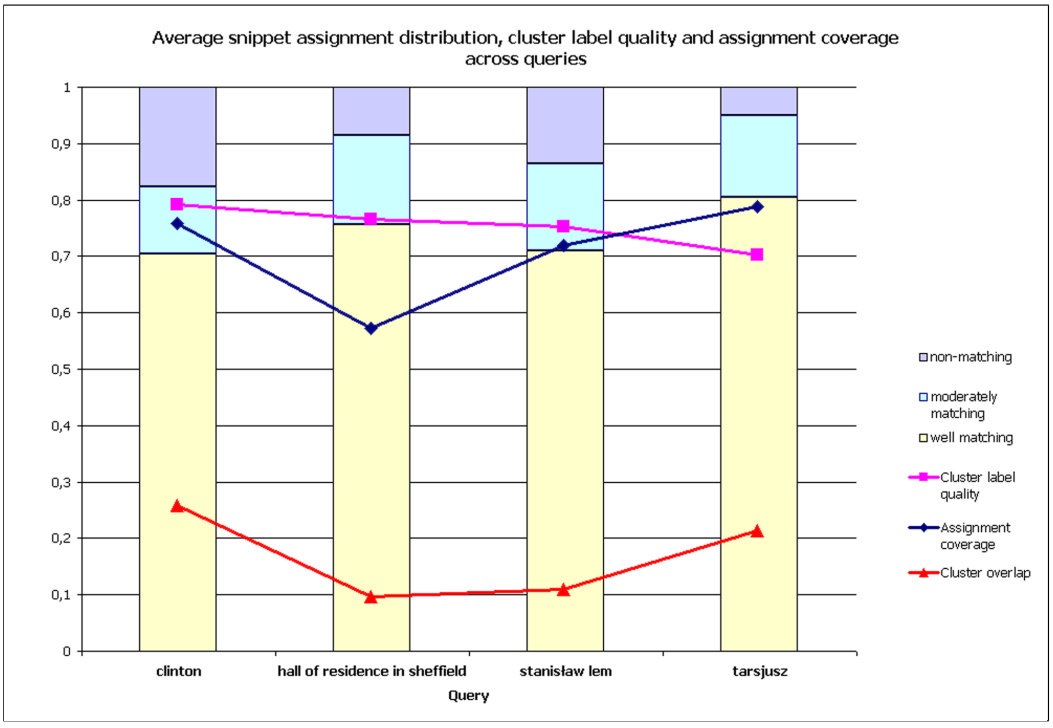
[38] Detailed results of our survey are provided on a CD accompanying this thesis, see Appendix A.

Nevertheless, several interesting observations as to the nature of search results clustering and user evaluation still can be made.

Evaluation results across queries

In Figure 5.2 we present snippet assignment distribution along with cluster label quality and snippet assignment coverage for each individual query. We used arithmetic averaging to aggregate the values over all evaluators.

Figure 5.2 *Evaluation results across queries*



The above chart reveals a number of interesting properties of the analysed groupings. First of all, varying from 80 up to 95%, the percentage of well- and moderately assigned snippets is relatively high. Although we consider this result as quite promising, we are aware that for certain "particularly difficult" queries these figures may turn out to be significantly worse.

As it can be observed, for general queries (i.e. *clinton* and *stanisław lem*) group content assignments are slightly less precise, which manifests in higher numbers of non-matching snippets. One of possible causes for this may be higher rates of cluster overlap for those queries. For the query *clinton*, for example, almost 30% of the snippets have been confined to more than one group. Consequently, due to the larger number of assignments, the probability of LINGO making a mistake in cluster content discovery is also higher.

Interesting is the connection between cluster label quality and assignment coverage. Surprisingly, values of these measures significantly differ for the two specific queries (i.e. *hall of residence in sheffield* and *tarsjusz*) we have had evaluated. The most natural explanation for this is that for a query with low assignment coverage, where a significant number of snippets is put in the "Other topics" group, the remaining documents contain no "outliers" and hence create coherent and useful groups. Likewise, for queries for which only a few

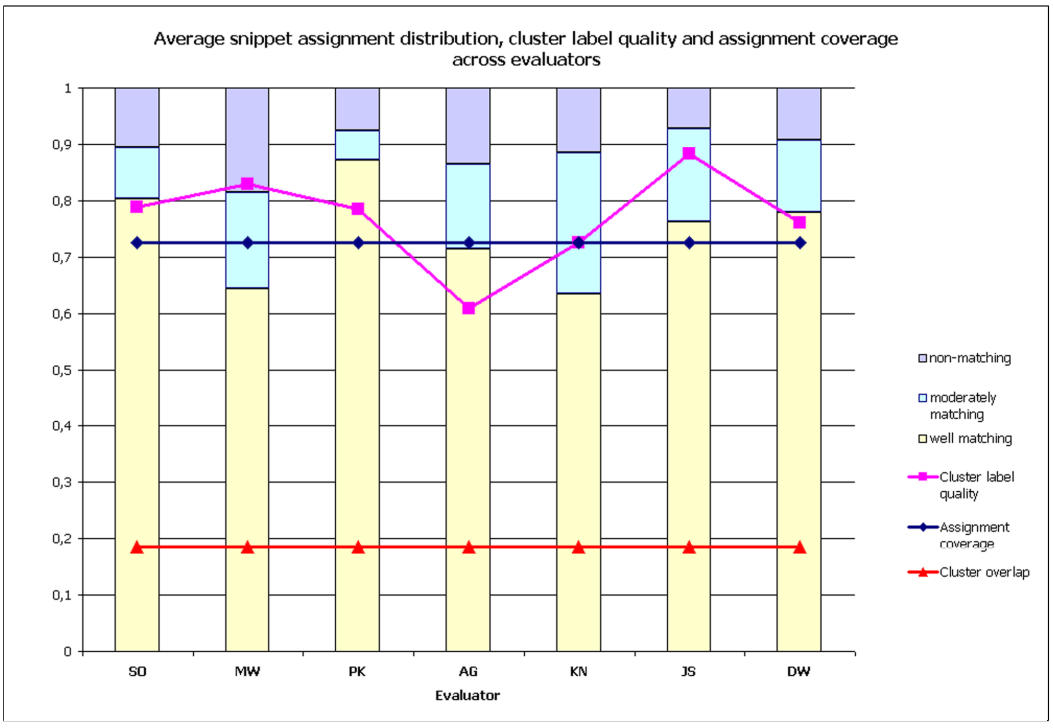
documents are placed in the "Other topics" cluster, not all useless documents may have been filtered out, which in turn may result in a greater number of unintuitive groups. In case of the query *tarsjusz* noteworthy is that judging many groups as useless (almost 30%) helps achieve high rates of precision, as it is assessed for useful groups only.

Finally, contrary to our intuition that, due to its great complexity, the Polish language should affect the quality of clustering, the queries *stanisław lem* and *tarsjusz* scored better than those in English. One of possible explanations for this might be that Polish was the native language for all of evaluators. Consequently, they might have felt more comfortable while assessing results in their native tongue, which could have contributed to the slight difference in overall results. Therefore, a general conclusion might be drawn that different nationalities and varied command of foreign languages among the evaluators should be taken into account when analysing user feedback. This conclusion, however, needs further verification on a larger set of queries and with a larger group of evaluators.

Evaluation results across evaluators

Figure 5.3 shows snippet assignment distribution, cluster label quality and snippet assignment coverage for particular evaluators. To aggregate the values for different queries arithmetic averaging was used. Snippet assignment coverage and cluster overlap remain constant as they do not depend on user choices.

Figure 5.3
Evaluation results across evaluators



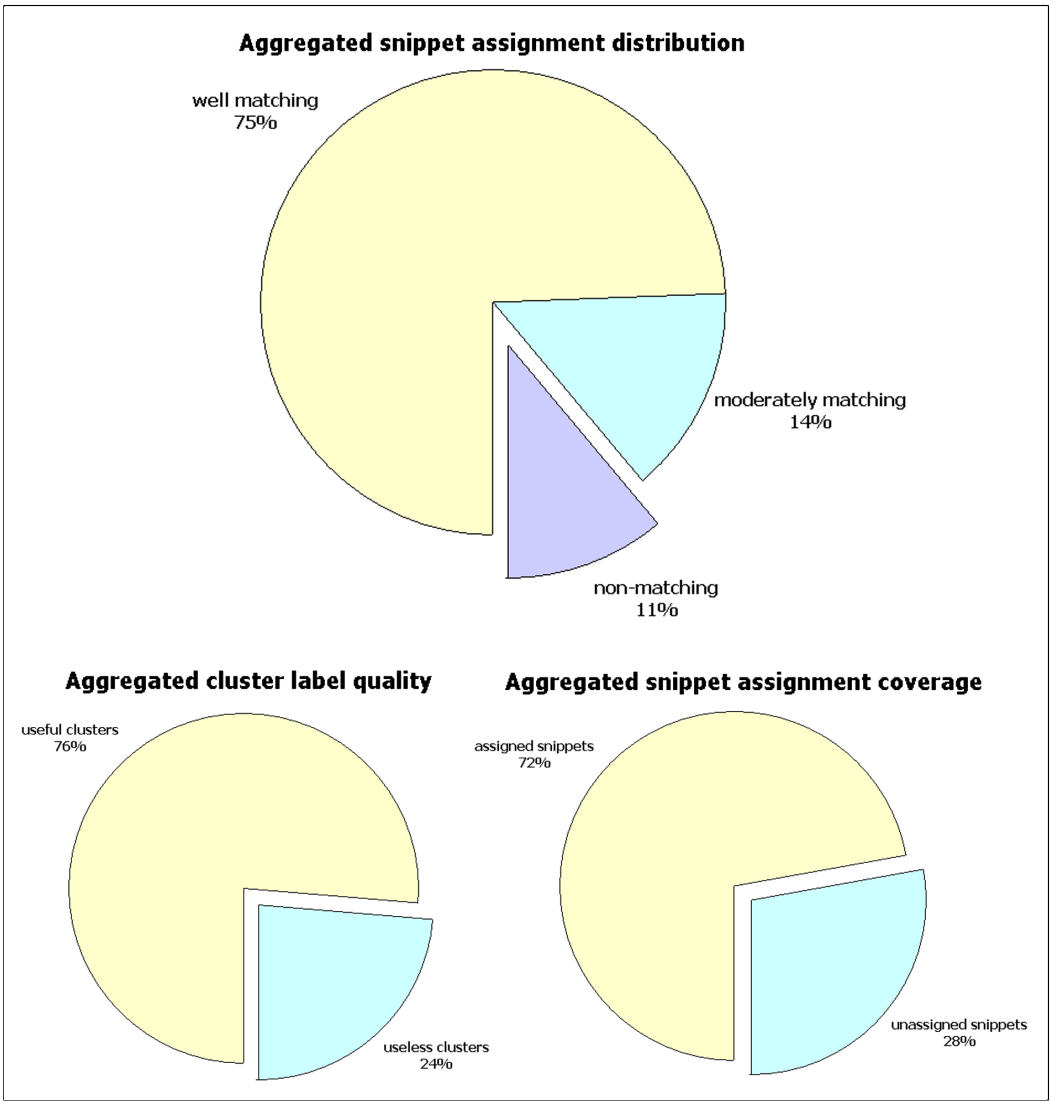
Results presented above clearly demonstrate how subjective are user relevance judgements. The percentage of well-matching snippets varies from over 60% in case of one user to almost 90% for another evaluator. Similarly, cluster label quality judgements change within the 20% range.

A closer analysis of individual evaluation forms reveals that certain cluster labels were marked as useless, even though they were grammatically correct and represented meaningful clusters for the majority of users. The reason for this may be that users who were less familiar with the idea of web search results clustering rejected all clusters that, according to their understanding of the query, should not have been created[39] – they did not *expect* them to be created. This is an excellent example of how the users' background knowledge influences their judgements. We conclude that this factor should be considered while deciding on the set of queries to be evaluated.

Aggregated evaluation results

As a summary of user evaluation of LINGO in Figure 5.4 we present the values of all measures averaged over both queries and evaluators.

Figure 5.4 *Aggregated evaluation results*



[39] Two examples of clusters which led to major disagreements among evaluators are "Elektroniczny Dziennik Ludowy" and "Zoologia Niezwykłe Superkompendium Wiedzy" for the query tarsjusz. Detailed user answers are provided in a CD accompanying this thesis, see Appendix A.

Being aware of the extremely small scale of our experiments, we only conclude that they have confirmed that the first-describe-then-cluster approach is capable of generating meaningful and reasonably described clusters. We feel that our algorithm needs further assessment which should concentrate primarily on the comparison to other methods such as Suffix Tree Clustering or Agglomerative Hierarchical Clustering.

Evaluation within Carrot²

The Carrot² framework seems a perfect environment for experiments on a larger scale as once the evaluation component is written, it can be used regardless of the actual clustering algorithm being assessed. Moreover, the on-line character of evaluation is likely to encourage more users to contribute their time and effort to the experiment.

5.3 Sample results

In this section we analyse the results we obtained from LINGO for a number of real-world queries and search engines. We have decided to test our algorithm on pairs of queries of different characteristics, e.g. a general versus a specific one, an English and a Polish one. This let us examine the influence of such factors as: the type and language of the query, the source search engine, the number of input snippets or the language identification on the overall quality of clustering. We also compare the results produced for the same set of input snippets by Vivisimo [Vivisimo, 02a] and by our algorithm.

Unless noted otherwise the clustering was performed in the following conditions:

Parameter	Value
Query language	English
Source search engine	Google
Algorithm	LINGO
Language identification	Yes
Snippet count	100
Candidate Label Threshold	0.775
Snippet Assignment Threshold	0.150
Term Frequency Threshold	2
Label Similarity Threshold	0.30

5.3.1 General and specific queries

As noted in [Weiss, 01], the quality of web search clustering is to a large extent influenced by the actual formulation of the input query. Most of the grouping algorithms are likely to perform better on general queries for which the spectrum of topics among the returned

snippets is fairly wide. Below we continue the "Clinton Example" started in [Zamir, 99] and further followed in [Weiss, 01].

Figure 5.5
Clustering results for a general and a specific query

	Left	**Right**
Query	*clinton*	*senator hillary rodham clinton*
Source search engine	Google	Google

As it might have been expected, LINGO performs better for queries that cover a larger number of subjects. For the general query (on the left), even though a larger number of clusters had been created, almost all labels indicate meaningful groups. In case of the specific query (on the right) however, several labels seem to give little clues as to the content of the cluster ("Calls", "Public", "Seven", "Senator Hillary Rodham Clinton John"). Additionally, for the specific query, a larger number of snippets had been confined to the "Other" group, which confirms that such queries are more "difficult" to cluster.

Noteworthy is that, in contrast to the previous "Clinton Example" experiments, only a single label ("Clinton Scandals") echoes the topic that would dominate the search results in 1999.

5.3.2 The influence of the number of snippets

The number of snippets processed for a single query is likely to have impact on two major aspects of the results: the quality of groups' description and the time spent on clustering. Having more input documents may eliminate some of the very specific phrases (which in

many cases turn out to be misleading) replacing them with their more general counterparts. Inevitably, the increased quality of description will be achieved at the cost of longer processing times (not only will the number of documents increase but also there will be more indexing terms). Below we compare clustering results for the same query but different numbers of input snippets.

Figure 5.6
Clustering results for 100 and 300 input snippets

	Left	**Right**
Query	*data mining*	*data mining*
Source search engine	Google	Google
Input snippets count	100	300

The most striking difference between the above groupings is that with the increase in the number of snippets, the number of groups is also significantly larger. The most obvious explanation for this lies in the fact that new input documents simply introduce new topics.

Indeed, the results on the right (300 snippets) contain a number (e.g. "Oracle9i Data Mining", "Microsoft SQL Server", "Case Studies") of groups absent from the grouping on the left. Closer examination of the "Oracle9i Data Mining" group reveals that all its members are in the third hundred of snippets and thus were not available in the 100-snippets setting.

The accuracy of cluster description seems to be better in the 300-snippets results. The "Untangling Text Data Mining" label is too specific, whereas its equivalent in the 300-snippets setting – "Text Data Mining" – is much better.A similar pair can be "Second International Conference" and "International Conference". The reason for this may be that in the 300-snippets grouping more documents match the discussed clusters somehow enforcing a more general description.

Finally, increasing the number of input snippets severely affects the processing times. In the 100-snippets setting the clustering process took mere 0.67 second, while processing of 300 input documents required almost ten times as much time – 6.09 sec.

5.3.3 Two different search engines

In web search results clustering, the fundamental assumption is that grouping algorithms work on the snippets returned by some web search engine. Consequently, the quality of clustering strongly depends on the quality of the underlying search service. Below, we present two groupings obtained from LINGO for the same query directed to two different search engines.

Figure 5.7
Clustering results for two different source search engines

	Left	**Right**
Query	*8051 programming*	*8051 programming*
Source search engine	Google	AllTheWeb

A closer analysis of the above results reveals that, despite being based on two potentially different sets of snippets, the groupings share a great deal of common topics: "Multitasking", "Customizing the Microcontroller", "Programming and Interfacing the Microcontroller", "Embedded Systems", "Training on Programming" and "Need some Help in Programming". This suggests that the two underlying search engines have a similar coverage of the Web and similar document ranking algorithms.

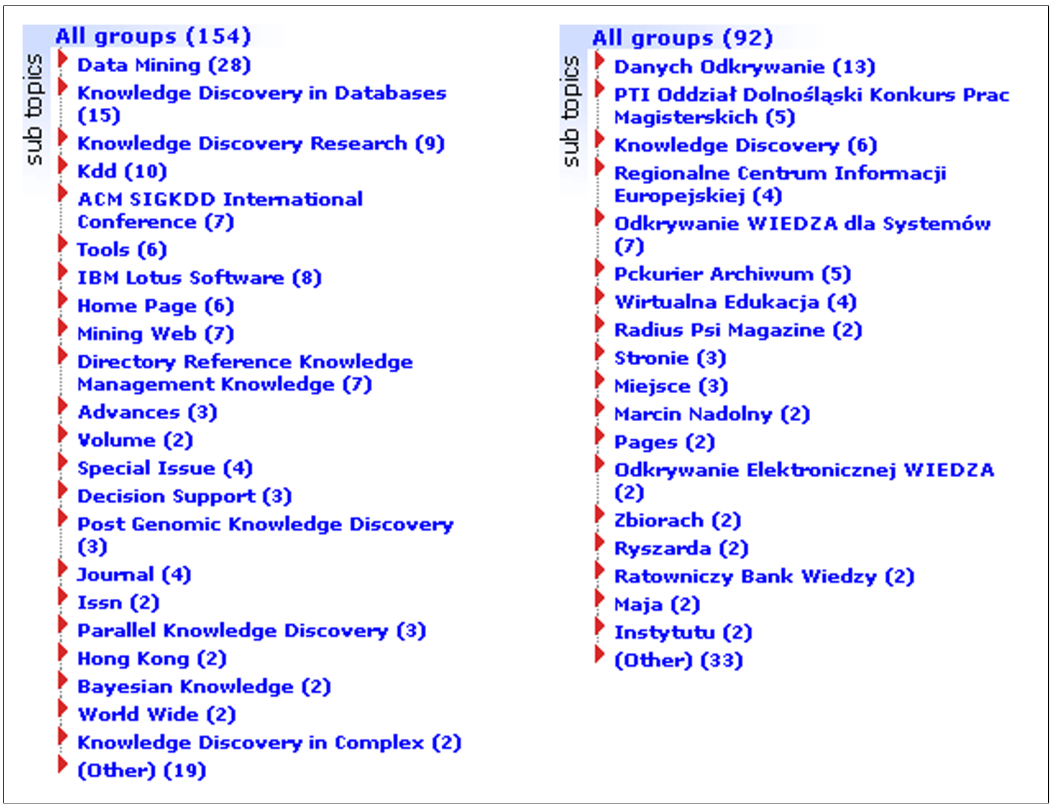
Remarks on LINGO's preprocessing strategy

One of the cluster descriptions generated from the Google's results (on the left) proves that LINGO's preprocessing algorithm is still far from perfection. In the "Books Gt" label, the "Gt" token comes from an HTML entity ">" and should have been definitely filtered out. Another observation is that being a part of the query, the token "8051" never appears in the cluster labels. This is caused by the digit elimination strategy that allows numbers only when accompanied by letters. Clearly, a better preprocessing algorithm would also take the *context* of the number into account. Finally, noteworthy is LINGO's (sometimes working…) word case preservation strategy which tries not to change the case of acronyms appearing in such labels as "AVR and Programming" or "QFP Programming Adapters".

5.3.4 Two different languages

Many of the clustering algorithms make an implicit assumption that they operate on texts written in English. Although over 50% of the Internet's documents are indeed in that language [WebLanguages, 02], we feel that other languages should not be neglected. That is why we have decided to test the performance of LINGO on two queries that mean the same, but are in two different languages.

Figure 5.8 *Clustering results for a query in English and Polish*



	Left	Right
Query	*knowledge discovery*	*odkrywanie wiedzy*
Query language	English	Polish
Source search engine	Google	Google

The most apparent difference between the above groupings is that the majority of English cluster labels form grammatically correct phrases while the Polish ones do not ("Odkrywanie Elektronicznej Wiedza" and "Odkrywanie Wiedza dla Systemów" are incorrect, "Danych Odkrywanie" seems awkward). The reason for this is LINGO's inability to restore the original inflected forms of the stemmed words in Polish. The issues of clustering Polish web search results are specifically addressed in [Weiss, 01].

5.3.5 The influence of the Candidate Label Threshold

One of the most important parameters of LINGO is the Candidate Label Threshold, which determines the number of clusters to be created (see Section 4.2.6 for details). Below we compare two groupings obtained for the same query but different values of the analysed parameter.

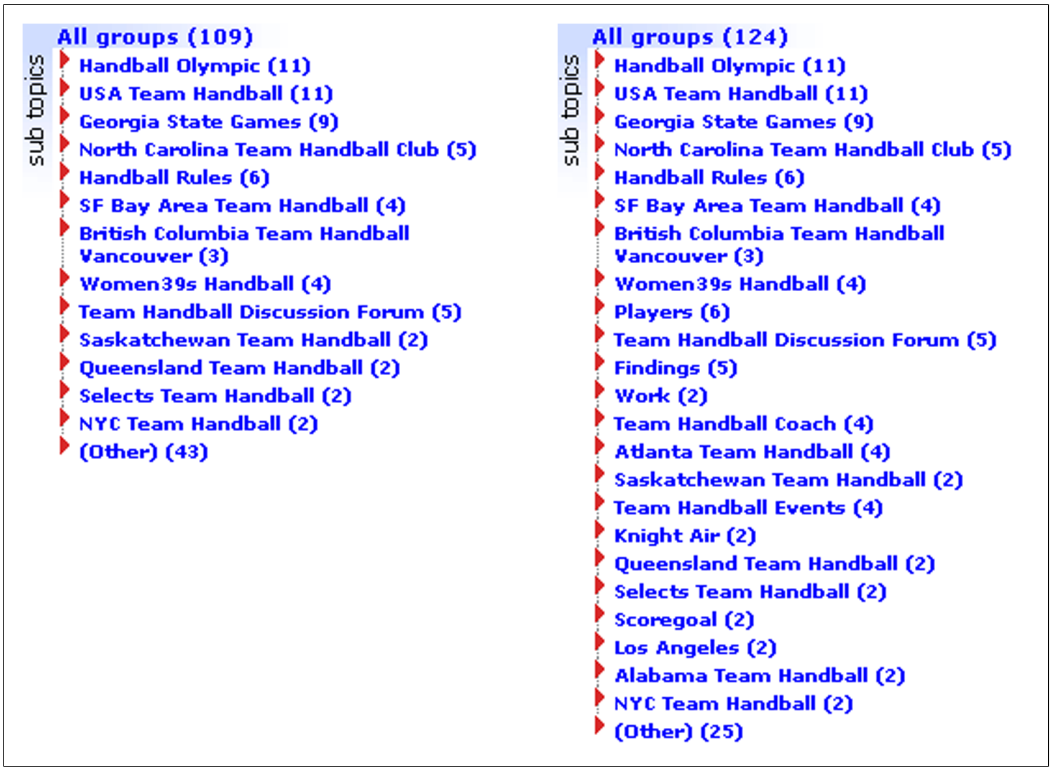


Figure 5.9 Clustering results depending on the Candidate Label Threshold

	Left	Right
Query	*team handball*	*team handball*
Source search engine	Google	Google
Candidate Label Threshold	0.70	0.80

Due to the characteristic of LINGO's label count calculation method (always a certain number of the first abstract concepts is considered – see Section 4.2.3.2 for details) the set of clusters on the left is exactly a subset of the set on the right. Therefore, while tuning the value of the Candidate Label Threshold a compromise must be sought between the coverage of topics and the number of useless cluster labels.

In the results on the left only one group seems to be useless ("Selects Team Handball"), while in the setting where more clusters are created the number of misleading labels is greater ("Work", "Findings", "Scoregoal"). However, a number of additional useful groups also appeared such as "Handball Rules", "Players" or "Alabama Team Handball".

5.3.6 The influence of language identification

In case of some queries, a search engine may return a mixture of snippets in several different languages. For such input, a clustering algorithm that assumes its input data to be in one language only (e.g. English) would inevitably recognise stopwords of other languages as frequent terms. As a consequence, the stopwords might form unintuitive or even useless clusters. Let us we compare two groupings produced by our algorithm with and without language identification step in the preprocessing phase.

Figure 5.10 *Clustering results for an algorithm with/without language identification*

	Left	**Right**
Query	*salsa*	*salsa*
Query language	English/Spanish	English/Spanish
Source search engine	Google	Google
Language identification	Yes	No (English only)

As it can be easily observed, the English-only version of LINGO (on the right) generated a large group of snippets described with a Spanish stopword "las". Another group – "En Salsa" – is also of little value, as the whole query was concerned with "Salsa" and the preposition "en" does not carry any new information. The multilingual version of our algorithm (on the left), by detecting the language of each snippet, can prevent any stopword[40] from appearing in cluster labels.

5.3.7 LINGO vs. Vivisimo

To our best knowledge, Vivisimo is the most mature web search results clustering engine available. The author of this thesis felt sorely tempted to compare the groupings obtained from the award-winning system and an absolute novice in the field. The test procedure was as follows. First, we issued a query *tarsjusz*[41] to the Vivisimo engine, saving the source input snippets. Next, we converted the snippets to the Carrot^2 XML format [Weiss, 03] and clustered them using LINGO. In this way both algorithms worked on the same input data. Below we present the outcome of the experiment.

Figure 5.11 *LINGO vs. Vivisimo*

	Left	**Right**
Query	*tarsjusz*	*tarsjusz*
Query language	Polish	Polish
Source search engine	Altavista, Netscape, MSN, LookSmart, Lycos, FindWhat	Altavista, Netscape, MSN, LookSmart, Lycos, FindWhat

[40] Any stopword in a language for which a stoplist is available.

[41] Tarsier (in Polish: *tarsjusz* or *wyrak*) is a type of small mammal with a round head and unusually large eyes

	Left	Right
Algorithm	Vivisimo	LINGO
Language identification	?	Yes

To our great satisfaction, both of the algorithms correctly identified the most important topics such as "Wislawa Szybmorska", "Nobel Prize for Literature", "Poezje wybrane" or "Svenska Akademien". Closer examination of the results reveals that there are two more common clusters "God Is My Co-Pilot Lyrics" (labelled "Lyrics" by LINGO) and "Opowieść o Agatce" (which LINGO misleadingly[42] described as "Mysz potrafi myśleć na Kim"). The general impression, however, is that LINGO still lacks the Vivisimo's precision in labelling clusters.

Does Vivisimo support overlapping clusters?

Interesting is the analysis of the number of snippets confined to each group in the above example. LINGO identified 7 documents about "Nobel Prize for Literature", while in the Vivisimo's results a similar group has only 3 members. Likewise, LINGO's "Svenska Akademien" contains 4 snippets, whereas the parallel group from Vivisimo is much smaller. When analysing the actual content of the clusters, the "missing" snippets were found in the "Press Releases" group. Apparently, Vivisimo does not support overlapping clusters which sometimes results in some groups being smaller than expected.

[42] The author of this thesis knows a person for whom the association between "Opowieść o Agatce" and "Mysz potrafi myśleć [na niebiesko]" is more than obvious though.

6 Conclusions and further work

Being a fairly young discipline of Computer Science, web search results clustering is gaining increasingly promising perspectives for the future. On the one hand, extensive research is being conducted on faster and more precise clustering algorithms, and on the other, the scientific advances are followed by practical implementations of the web search clustering idea, some of which, such as Vivisimo, have already achieved commercial success.

With this thesis we aimed to contribute to both the scientific and the practical trend in web search clustering. Analysing the currently available algorithms, we observed that little emphasis was being placed on the quality of thematic groups' description. This was the major incentive for us to propose and implement a new method whose main priority would be to provide concise and meaningful group labels.

Inspiration for LINGO

During the course of writing this thesis we reviewed recent literature on automatic text processing techniques, placing special emphasis on the advances in document clustering. The direct inspiration for our work was a clustering algorithm called SHOC – Semantic Hierarchical On-line Clustering. It is this approach that turned our attention to Latent Semantic Indexing and Singular Value Decomposition. To understand fully the two concepts we had to include the problems of linear algebra in our literature studies.

The main result of this project is the design of LINGO – a description-oriented web search results clustering algorithm. Following advice of creators of Vivisimo, we have decided to adopt an unconventional approach in which good cluster labels are identified first and then, based on them, the actual assignments of search results to groups are made. We strongly felt that it should be easier to find documents that match a meaningful cluster label than to describe a potentially senseless group of snippets.

To verify the practical value of our idea we implemented LINGO as a component of the Carrot^2 framework. During the implementation work, we identified additional factors that significantly influence the quality of clustering. We claim that proper preprocessing of the input data, including language identification, is of crucial importance in web mining tasks. A similar observation has been made by other researches [Stefanowski and Weiss, 03b].

We also proposed an expert-based scheme for evaluation of web search results clustering. The scheme aims at assessing both the individual groups and the set of clusters as a whole. Due to the limited amount of time allocated to this project, we managed to collect only a few surveys regarding LINGO's performance in real-world tasks. However, according to the feedback we received and to our own analyses, we feel that the main goal of this thesis has been attained – LINGO produces reasonably described and meaningful clusters. This

confirms our initial intuition as to the the practical usefulness of the first-describe-then-cluster concept.

Finally, we would like to emphasise that by integrating our algorithm into the Carrot^2 framework not only were we able to carry out experiments with real-world data, but also we could contribute to a worthwhile Open Source initiative. The prospect of LINGO being still worked on and perhaps one day used as a part of a full-blown search service makes this work particularly rewarding.

6.1 Scientific contributions

The scientific contributions of this work are as follows:

a. **Design of a description-oriented clustering algorithm**

We have proposed a novel web search results clustering algorithm in which strong emphasis is placed on the high quality of group descriptions. To our best knowledge, LINGO is the first published algorithm[43] in which the first-describe-then-cluster approach has been successfully used. Additionally, we have identified other factors that are crucial for the high quality grouping. These are the careful preprocessing of the input data and the snippet language identification.

b. **Open Source implementation of LINGO**

We have implemented our proposed clustering method and released the code as a part of the Open Source Carrot^2 framework. Thus, the reference realisation of LINGO can serve as an environment for further experiments with e.g. the applications of linear algebra to document clustering.

c. **Expert-based approach to clustering evaluation**

We proposed an evaluation method that adapts the standard Information Retrieval metrics to the problem of clustering results assessment. Being able to express the inherently ambiguous human relevance judgements by means of standardised IR measures, our approach makes it possible to compare different clustering algorithms as well as to measure the performance of an individual method. We used our proposed assessment scheme to verify the practical performance of LINGO.

The limited user evaluation of LINGO which we managed to carry out demonstrated its practical usefulness. On average, over 77% of groups discovered by LINGO have been judged as useful. Additionally, over 88% of documents in clusters have been judged as well- or moderately-matching.

[43] Vivisimo also produces remarkably well described clusters, but the details of the algorithm have not been revealed.

6.2 Further work

In this thesis we have proposed a novel approach to web search results clustering, which turned out to be fairly effective in practical applications. Nevertheless, there are still many problems which we did not manage to address. Below we present a list of research directions that we feel are worth following.

a. **Hierarchical clustering**

In its present form LINGO is unable to generate a hierarchical structure of clusters. In some cases this may cause our algorithm to generate groups that are too large or too general. As we argue in Section 4.6, Latent Semantic Indexing is potentially capable of capturing the semantic associations between cluster labels, which may be used in the cluster merging step.

b. **Improvements in LINGO**

Apart from support for hierarchical clustering, the design of our algorithm can be improved in several other areas. Among them are more precise cluster label generation, better group content discovery and incremental processing. For the details we refer the Reader to Section 4.6.

For the time being, the implementation of LINGO is fairly computationally expensive. To alleviate the performance problems, in the first place faster SVD algorithms, such as this in [Drineas et al., 99], must be sought. Although to a lesser extent, specialised suffix sorting algorithms should also improve the running times of our algorithm. Finally, an implementation of the more robust trigam-based language identification scheme may be considered.

c. **The role of preprocessing**

During the course of implementing LINGO we observed the enormous influence of the preprocessing phase on the overall quality of clustering. As we have shown, especially in case of web snippets, simple character-based filtering methods often fail. It seems that more advanced context-aware, e.g. rule-based, preprocessing techniques could further improve the quality of grouping.

d. **Experiments on a larger scale**

Owing to the limited amount of time allocated to this project, we were unable to conduct detailed experiments on the design and real-world performance of our algorithm. First of all, the influence of design choices as to e.g. the term weighting scheme or the vector distance measure used in LINGO needs thorough examination. Secondly, a larger-scale user evaluation of the results produced by our algorithm could further confirm its practical value.

Bibliography

[AllTheWeb, 03]

AllTheWeb search engine. http://www.alltheweb.com.

[AllTheWeb FTP, 03]

FTP search engine. http://www.alltheweb.com/?cat=ftp.

[Ando, 00]

Rie Kubota Ando. *Latent Semantic Space: Iterative Scaling Improves Precision of Inter-document Similarity Measurement*.

[AnswerBus, 03]

A Web-based Question Answering System. http://www.answerbus.com.

[AltaVista Prisma, 03]

AltaVista Prisma search engine. http://www.altavista.com/prisma.

[Baeza-Yates et al., 99]

Richardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. ACM Press. 0-201-39829-X.

[Berry et al., 95]

Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien. *Using Linear Algebra for Intelligent Information Retrieval*. SIAM Rev., 37, pp. 537-595.

[Berry et al., 99]

Michael W. Berry, Zlatko Drmac, and Elizabeth R. Jessup. *Matrices, Vector Spaces, and Information Retrieval*. SIAM Rev. 41, pp. 335-362.

[Bharat and Henzinger, 98]

Krishna Bharat and Monika R. Henzinger. *Algorithms for Topic Distillation in a Hyperlinked Environment*. Proceedings of the 21st ACM SIGIR Conference on Research and Development in Information Retrieval, 1998.

[Burrows and Wheeler, 94]

M. Burrows and D. Wheeler. *A block sorting losseless data compression algorithm*. Technical Report 124, Digital Equipment Corporation, 1994.

[CiteSeer, 03]

Scientific Literature Digital Library. http://citeseer.nj.nec.com.

[Cormen et al., 90]

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The Massachussetts Institute of Technology, 1990.

[Deerwester et al., 90]

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. *Indexing by Latent Semantic Analysis*. J. American Society for Information Science, 41 (1990), pp. 391-407.

[Dogpile, 03]

Dogpile meta-search engine. http://www.dogpile.com.

[Drineas et al., 99]

P. Drineas, Alan Frieze, Ravi Kannan, Santosh Vempala, and V Vinay. *Clustering in large graphs and matrices*. Proceedings of ACM-SIAM Symposium on Discrete Algorithms, 1999.

[Froogle, 03]

Web shopping search engine. http://www.froogle.com.

[Google, 03]

Google search engine. http://www.google.com.

[Google, 03b]

Spelling correction of the query "Britney Spears". http://www.google.com/jobs/britney.html.

[Google Groups, 03]

Usenet search engine. http://groups.google.com.

[Grefenstette, 95]

Gregory Grefenstette. *Comparing two language identification schemes*. 3rd International Conference on Statistical Analysis of Textual Data, Rome, 1995.

[Hartfiel and Hobbs, 87]

D. J. Hartfiel and Arthur M. Hobbs. *Elementary Linear Algebra*. Prindle, Weber & Shmidt. Boston. 0-87150-038-8.

[Hearst, 98]

M. A. Hearst. *The use of categories and clusters in information access interfaces*. In T. Strzalkowski (ed.), Natural Language Information Retrieval, Kluwer Academic Publishers, 1998.

[Hearst and Pedersen, 96]

M. A. Hearst and J. O. Pedersen. *Reexamining the Cluster Hypothesis: Scatter/Gather on Retrieval Results*. Proceedings of the Nineteenth Annual International ACM SIGIR Conference, Zurich, June 1996.

[HotSpot, 03]

Java HotSpot Technology. http://java.sun.com/products/hotspot/.

[iBoogie, 03]

iBoogie – meta search engine with automatic document clustering. http://www.iboogie.tv/.

[JAMA, 03]

The Java Matrix Package. http://math.nist.gov/javanumerics/jama/.

[Kartoo, 03]

Kartoo meta seach engine. http://www.kartoo.com.

[Karypis and Han, 00]

Yiming Yang and Jan O. Pedersen. *Concept Indexing: A Fast Dimensionality Reduction Algorithm with Applications to Document Retrieval and Categorization*. Technical Report TR-00-0016, University of Minnesota, 2000. http://citeseer.nj.nec.com/karypis00concept.html.

[KillerInfo, 03]

KillerInfo search engine. http://www.killerinfo.com/.

[Kleinberg, 98]

Jon Kleinberg. *Authoritative sources in a hyperlinked environment*. ACM-SIAM Symposium on Discrete Algorithms, pp. 668-667, San Francisco, USA, 1998.

[Kontostathis and Pottenger, 02]

April Kontostathis and William Pottenger. *A Mathematical View of Latent Semantic Indexing: Tracing Term Co-ocurrences*. http://www.cse.lehigh.edu/techreports/2002/LU-CSE-02-006.pdf.

[Kurcz et al., 90]

I. Kurcz, A. Lewicki, J. Sambor, K. Szafran, and J. Worończak. *Słownik frekwencyjny polszczyzny współczesnej, t. 1-2*. Instytut Języka Polskiego PAN, Kraków, 1990.

[Landauer and Dumais, 96]

Michael L. Littman and Susan T. Dumais. *Latens Semantic Analysis and the measurement of knowledge*. Proceedings of the First Educational Testing Service Conference on Applications of Natural Language Processing in Assessment and Education, 1994.

[Landauer et al., 98]

Thomas K. Landauer, Peter W. Foltz, and Darrel Laham. *Introdction to Latent Semantix Analysis*. Discourse Processes, 25, pp. 259-284, 1998.

[Larsson, 98]

N. Jesper Larsson. *Notes on suffix sorting*. http://citeseer.nj.nec.com/larsson98notes.html.

[Larsson and Sadakane, 99]

N. Jesper Larsson and Kunihiko Sadakane. *Faster Suffix Sorting*. http://citeseer.nj.nec.com/larsson99faster.html.

[Lerman, 99]

Kristina Lerman. *Document Clustering in Reduced Dimension Vector Space*. unpublished.

[Littman et al., 96]

Michael L. Littman, Susan T. Dumais, and Thomas K. Landauer. *Automatic Cross-Language Information Retrieval using Latent Semantic Indexing*. Proceedings of SIGIR-96, Zurich, 1996.

[LookSmart, 03]

A human-made web directory. http://www.looksmart.com.

[Mamma, 03]

Mamma meta-search engine. http://www.mamma.com.

[Manber and Myers, 94]

U. Manber and G. Myers. *Suffix arrays: a new method for on-line string searches*. SIAM Journal of Computing, 22(5), pp. 953-948, 1993.

[Manzini and Ferragina, 02]

Giovanni Manzini and Paolo Ferragina. *Engineering a Lightweight Suffix Array Construction Algorithm*. ESA 2002, LNCS 2461, pp. 698-710, 2002.

[MapNet, 03]

MapNet meta search engine. http://maps.map.net/.

[Metacrawler, 03]

Metacrawler meta-search engine. http://www.metacrawler.com.

[Nicholson, 86]

Keith W. Nicholson. *Elementary Linear Algebra with Applications*. Prindle, Weber & Shmidt. Boston. 0-87150-902-4.

[ODP, 03]

Open Directory Project – a human-made web directory. http://dmoz.org.

[Opial, 69]

Zdzisław Opial. *Algebra Wyższa*. Polskie Wydawnictwo Naukowe. Warszawa.

[Page et al., 98]

L. Page, S. Brin, S. Motwani, and R. Winograd. *The Page Rank citation ranking: Bringing order to the Web*. Technical Report, Stanford University, 1998.

[Porter, 80]

M. F. Porter. *An algorithm for suffix stripping*. Program, 14 (3), pp. 130-137, 1980.

[Questlink, 03]

Electronic parts search engine. http://www.questlink.com.

[REP, 03]

Robots Exclusion Protocol. http://www.robotstxt.org/.

[Riboni, 02]

Daniele Riboni. *Feature Selection for Web Page Classification*. http://citeseer.nj.nec.com/554644.html.

[Salton, 89]

Gerald Salton. *Automatic Text Processing*. Addison-Wesley. 0-201-12227-8.

[Selberg, 99]

Eric W. Selberg. *Towards Comprehensive Web Search*. Doctoral Dissertation, University of Washington, 1999.

[Shenker et al., 01]

Adam Shenker, Mark Last, and Abraham Kandel. *Design and Implementation of a Web Mining System for Organizing Search Engine Results*. Proceedings of the CAiSE'01 Workshop Data Integration over the Web (DIWeb'01), pp. 62 - 75, Interlaken, Switzerland, 2001.

[Sherman, 02]

Chris Sherman. *Anatomy of a Search Engine: Inside Google*. http://www.searchenginewatch.com/searchday/article.php/2161091.

[START, 03]

Natural Language Question Answering System. http://www.ai.mit.edu/projects/infolab/.

[Stefanowski and Weiss, 03]

Jerzy Stefanowski and Dawid Weiss. *Carrot^2 and Language Properties in Web Search Results Clustering*. Proceedings of the First International Atlantic Web Intelligence Conference, Madrit, Spain, vol. 2663, 2003, pp. 240-249.

[Stefanowski and Weiss, 03b]

Jerzy Stefanowski and Dawid Weiss. *Web search results clustering in Polish: experimental evaluation of Carrot*. Advances in Soft Computing, Intelligent Information Processing and Web Mining, Proceedings of the International IIS: IIPWM´03 Conference, Zakopane, Poland, vol. 579 (XIV), 2003, pp. 209-22.

[Stern, 02]

Ulrich Stern. *Java vs. C++*. http://verify.stanford.edu/uli/java_cpp.html.

[Sullivan, 03]

Danny Sullivan. *Web Directory Sizes*. http://www.searchenginewatch.com/reports/article.php/2156411.

[Teoma, 03a]

Teoma search engine. http://www.teoma.com.

[Teoma, 03b]

Teoma Technology Overview. http://sp.teoma.com/docs/teoma/about/searchwithauthority.html.

[Ukkonen, 95]

Esko Ukkonen. *On-line construction of suffix trees*. Algorithmica 14, no. 3, pp. 249-260, 1995.

[Vivisimo, 02a]

Vivisimo Clustering Engine. http://vivisimo.com, 2002.

[Vivisimo, 02b]

Vivisimo Clustering Engine - Techonology Overview. http://vivisimo.com/faq/Technology.html, 2002.

[WebLanguages, 02]

Distribution of languages on the internet. http://www.netz-tipp.de/languages.html.

[Weiss, 01]

Dawid Weiss. *A Clustering Interface for Web Search Results in Polish and English*. Master Thesis, Poznan University of Technology, 2001.

[Weiss, 03]

Dawid Weiss. *Carrot^2 Developers Guide*. http://www.cs.put.poznan.pl/dweiss/carrot/site/developers/manual/manual.pdf.

[Weiss, 03b]

Dawid Weiss. *A free stemming engine for the Polish language*. http://www.cs.put.poznan.pl/dweiss/index.php/projects/lametyzator/index.xml?lang=en.

[WiseNut, 03]

WiseNut search engine. http://www.wisenut.com/.

[Wroblewski, 03]

Michał Wróblewski. *Hierarchical Web documents clustering algorithm based on the Vector Space Model*. Master Thesis, Poznan University of Technology, 2003.

[Yahoo, 03]

Yahoo search engine. http://www.yahoo.com.

[Yang and Pedersen, 97]

Yiming Yang and Jan O. Pedersen. *A Comparative Study on Feature Selection in Text Categorization*. Proceedings of ICML-97, 14th International Conference on Machine Learning. Morgan Kaufmann Publishers, San Francisco, US, 1997, pp. 412-420.

[Yuwono and Lee, 96]

B. Yuwono and D. L. Lee. *Search and ranking algorithms for locating resources in World Wide Web*. Proceedings of the International Conference on Data Engineering (ICDE), pp. 164-171, New Orleans, USA, 1996.

[Zamir and Etzioni, 98]

Oren Zamir and Oren Etzioni. *Document Clustering: A Feasibility Demonstration*. Proceedings of the 19th International ACM SIGIR Conference on Research and Development of Information Retrieval, 1998, pp 46-54.

[Zamir and Etzioni, 99]

Oren Zamir and Oren Etzioni. *Grouper: A Dynamic Clustering Interface to Web Search Results*. WWW8/Computer Networks, Amsterdam, Netherlands, 1999.

[Zamir, 99]

Oren E. Zamir. *Clustering Web Documents: A Phrase-Based Method for Grouping Search Engine Results*. Doctoral Dissertation, University of Washington, 1999.

[Zha, 98]

Hongyuan Zha. *A subspace-based model for information retrieval with applications in Latent Semantic Indexing*. Proceedings of Irregular '98 Lecture Notes in Computer Science, 1457, (Springer-Verlag, 1998), pp. 29-42.

[Zhang and Dong, 01]

Dell Zhang and Yisheng Dong. *Semantic, Hierarchical, Online Clustering of Web Search Results*. Accepted by 3rd International Workshop on Web information and data management, Atlanta, Georgia.

A Thesis CD contents

On a CD accompanying this thesis we provide the latest source code of LINGO and other Carrot^2 components along with digital versions of some of the papers referenced in this work. Below we describe the CD's contents in detail:

Table A.1 *Thesis CD contents*

Directory	Contents
/Carrot	A full checkout of the Carrot^2 CVS repository (dated June 19, 2003). Contains the full source code of LINGO and other Carrot^2 components. Refer to [Weiss, 03] for instructions on building and running the software.
/Evaluation	The Excel forms used in the process of user-based evaluation of LINGO. We also provide the seven questionaires filled in by users and the aggregated results sheet.
/Papers	All papers referenced in this thesis
/Thesis/Print	The printable version of this thesis in PDF format.
/Thesis/Source	Source version of this thesis in XML format (DocBook DTD). Stylesheets used to generate the output PDF file are also included.
/Thesis/Stylesheet	XSLT stylesheets used to generate the output PDF file with this thesis. The stylesheets are based on the DocBook XSLT developed by Norman Walsh. Please note that in their present form our stylesheets are extremely crude and should rather be treated as a temporary solution to DocBook formatting.

B Linear algebra terms

Below, we provide a brief glossary of linear algebra terms used throughout this thesis. For in-depth coverage of the topic we refer the Reader to [Nicholson, 86], [Hartfiel and Hobbs, 87] or [Opial, 69].

Length of a vector

Length $||\mathbf{v}||$ of a t-dimensional vector $\mathbf{v}$ can be computed using the following formula:

$$\|\mathbf{v}\| = \sqrt{\sum_{i=1}^{t} v_t^2}$$

Dot product (scalar product)

Dot product $\mathbf{u} \cdot \mathbf{v}$ of vectors $\mathbf{u}$ and $\mathbf{v}$ is defined as follows:

$$\mathbf{u} \cdot \mathbf{v} = \begin{cases} \|\mathbf{u}\|\|\mathbf{v}\|\cos\theta & \text{if } \mathbf{u} \neq \mathbf{0} \text{ and } \mathbf{v} \neq \mathbf{0} \\ 0 & \text{otherwise} \end{cases}$$

where θ is the angle between $\mathbf{u}$ and $\mathbf{v}$. Note that $\mathbf{u} \cdot \mathbf{v}$ is a *number* even though $\mathbf{u}$ and $\mathbf{v}$ are vectors. For this reason $\mathbf{u} \cdot \mathbf{v}$ is sometimes called a **scalar product** of $\mathbf{u}$ and $\mathbf{v}$.

Orthogonal vectors

Two nonzero vectors $\mathbf{v}$ and $\mathbf{u}$ are orthogonal if and only if $\mathbf{u} \cdot \mathbf{v} = 0$.

Linear combination

For a set of n vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots \mathbf{v}_n$ in a vector space V, a sum of scalar multiples of these vectors

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots a_n\mathbf{v}_n$$

is called **linear combination of the vectors** $\mathbf{v}_1, \mathbf{v}_2, \ldots \mathbf{v}_n$.

Span of a set of vectors

The set of all linear combinations of a set $\{\mathbf{v}_1, \mathbf{v}_2, \ldots \mathbf{v}_n\}$ of vectors in a vector space V is called its **span** and is denoted by

$$\text{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots \mathbf{v}_n\}$$

Linear independence

A set $\{\mathbf{v}_1, \mathbf{v}_2, \ldots \mathbf{v}_n\}$ of vectors is **liearly independent** if it satisfies the following condition:

If $s_1\mathbf{v}_1 + s_2\mathbf{v}_2 + \ldots s_n\mathbf{v}_n = \mathbf{0}$, then $s_1 = s_2 = \ldots = s_n = 0$

Basis of a vector space

A set $\{\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_n\}$ of vectors in a vector space V is called a **basis** of V if it satisfies the following conditions:

(1) $\{\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_n\}$ is linearly independent

(2) $V = \text{span}\{\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_n\}$

A basis whose vectors are pairwise orthogonal is called an **orthogonal basis**. Additionally, if all basis vectors are of unit length then the basis is call an **orthonormal basis**.

Dimension of a vector space

If $\{\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_n\}$ is a basis of the vector space V, the number n of vectors in the basis is called the **dimension** of V, and we write

$\dim V = n$

Row and column space of a matrix

If A is an $m \times n$ matrix, the vector space spanned by its m rows is called the **row space** of A and is denoted as row(A). Similarly, the vector space spanned by the n columns of matrix A is called the **column space** of A and is denoted as col(A).

Rank of a matrix

The dimension of the row (or column) space of a matrix A is called the **rank** of A and is denoted rank A.

Orthogonal matrix

A square matrix A is orthogonal when its colums (or rows) are pairwise orthogonal.

Frobenius norm of a matrix

The Frobenius norm of an $m \times n$ matrix A is defined as follows:

$$\|A\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij}}$$