A HIERARCHICAL WWW PAGES CLUSTERING ALGORITHM BASED ON THE VECTOR SPACE MODEL

by
MICHAŁ WRÓBLEWSKI

Supervisor
JERZY STEFANOWSKI, Assistant Professor

Referee
ROBERT SUSMAGA, Assistant Professor

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

July 2003

HIERARCHICZNY ALGORYTM GRUPOWANIA STRON WWW WYKORZYSTUJĄCY MODEL PRZESTRZENI WEKTOROWEJ

MICHAŁ WRÓBLEWSKI

Promotor
dr hab. inż. JERZY STEFANOWSKI

Recenzent
dr inż. ROBERT SUSMAGA

PRACA MAGISTERSKA

Specjalność Inżynieria Oprogramowania, Instytut Informatyki
Politechnika Poznańska, Poznań

lipiec 2003

ABSTRACT

This master thesis examines possibility of adaptation of hierarchical cluster analysis algorithm AHC (Agglomerative Hierarchical Clustering) based on the vector space model to Web search results clustering. We show here problems that arise when trying to use this algorithm in this application domain, where it must process textual data sets, containing very limited information.

We present here, how these problems may be overcome by using adequate techniques of pre-processing (use of phrases in the vector space model) and post-processing. In particular, we propose several methods of pruning of output dendrogram created by AHC, based on descriptions of clusters being elements of that tree.

In order to perform a verification of presented version of the algorithm, it has been implemented as a module of the Carrot^2 system. Moreover, we have carried out an evaluation of quality of results given by the algorithm.

STRESZCZENIE

Niniejsza praca magisterska bada możliwość adaptacji hierarchicznego algorytmu analizy skupień AHC (Agglomerative Hierarchical Clustering) opartego na modelu przestrzeni wektorowej do grupowania wyników zapytań do serwisów internetowych. Pokazujemy tu problemy, jakie powstają w momencie próby użycia tego algorytmu do tego typu zastosowań, gdzie musi on działać na tekstowych zbiorach danych, zawierających bardzo ograniczone informacje.

Przedstawiamy tu, jak można przezwyciężyć te problemy za pomocą odpowiednich technik wstępnego przetwarzania danych wejściowych (zastosowanie fraz w modelu przestrzeni wektorowej) i końcowego przetwarzania wyników algorytmu. W szczególności proponujemy kilka technik redukcji zbędnych węzłów wyjściowego dendrogramu tworzonego przez AHC w oparciu o opisy grup będących jego elementami.

W celu dokonania praktycznej weryfikacji przedstawionej przez nas wersji algorytmu, została ona zaimplementowana jako moduł systemu Carrot^2. Ponadto przeprowadzono też eksperyment mający na celu ocenę jakości wyników tworzonych przez algorytm.

ACKNOWLEDGMENTS

In the course of creating this thesis, I have come upon many difficulties, which I couldn't certainly solve easily or even at all using just my own skills. Finding solutions to these problems was possible only thanks to help from other persons. First of all, I would like to mention Mr. Jerzy Stefanowski, to whom I am deeply grateful for arousing my interest in Web search results clustering and important scientific suggestions. I give thanks also to Mr. Dawid Weiss, author and manager of the Carrot^2 project, whose help in solving technical problem was invaluable. Without his hard work on the Carrot^2, and earlier also on the Carrot system, this thesis would either simply not exist or it would take incomparably much more effort to complete it.

I would like here to express my gratitude to other Carrot^2 developers – Paweł Kowalik and Stanisław Osiński, with whom I had the pleasure to cooperate for a really long year. Working together, we have participated in creation of a really interesting system, with great research perspectives for the future.

I wish to thank people, whose time and effort have made evaluation of the results given by the algorithm possible – Jacek Błaszczyk, Irmina Masłowska, Stanisław Osiński, Jerzy Stefanowski and Dawid Weiss. Mrs. Irmina Masłowska deserves particular thanks for her valuable remarks on hierarchical results quality measures and giving me the ability to compare results of AHC and her clustering method.

Finally, I would like to express how important for me was the help of all the nearest people, who supported me during course of this thesis. It is hard for me to say, how much I owe you.

CONTENTS

1. INTRODUCTION

1.1. MOTIVATIONS

From the beginnings of civilization, man has always gathered knowledge. First (and the most important) invention in this area was writing, which enabled persistent storage of information. In ancient times and then Middle Ages great libraries like ones in Alexandria or Neneveh, storing thousands of books, were often the most important centers of civilization. When computers, and then the Internet have appeared, our abilities of gathering information have dramatically increased. The number of documents available in Internet grows exponentially since its origin. This is mostly due to the fact that automatic generation of Web documents is possible. Databases of world's currently largest Web search service, Google contains now about 3 *billion* of indexed Web pages, while just two years ago it contained "only" 1.3 billion [Google]. And even the biggest search services cover only a dozen or so percent of the Internet.

Moreover, the accessibility of information has considerably improved. Nowadays, one does not longer need to travel to Athens or Alexandria to find needed information – all you need to do is just to get on the Internet, type in a few words, click something and wait a few seconds for the result (at least so would claim the marketing wizards of the Internet search services).

Unfortunately, as the size of stored data grows, it is getting even harder and harder to find the relevant information that one is looking for. In a small library the librarian may know where to find each book, but dealing with quantity of documents such as mentioned in the first paragraph is beyond human capabilities. Moreover, as almost anyone can publish almost anything in the Web (which is therefore often concerned as a "Big Mess"), only a small part of information contained there is really *valuable*. This creates further challenge as user should get both *all* and *only* the information relevant to his query.

All above problems can be formulated as a single task of finding the set of documents on the Web relevant to a given user query. This is so called **Web Search Problem** [Selberg 99]. As a response to it, a special kind of Web service – **search engines** were created, which provide some – but still far from perfect – solution. Search engines are now among most popular net services, and are often points from which user starts to surf the Internet, so that one may call them "gates to the Web". A survey made recently by Search Engine Watch Web site revealed that they are the top information resource that Americans use when seeking answers (used 32 percent of the time) [SearchEngineWatch]. The most commonly used services are nowadays Yahoo [Yahoo], MSN [MSN], AltaVista [AltaVista] and Google [Google], receiving 200 millions queries daily and considered usually as the best one available.

In such typical service, the user types in his query, describing what she or he is looking for and the service returns a list of documents more or less relevant to this query. Although

algorithms and techniques used for finding this data may be different, user interaction in most of today search engines is based on two main models: query list (for the input) and ranked list (for the output). Unfortunately, it is still not possible to ask most of the services questions in the natural language (and it is even less probable to get a valid answer to such type of query). So the **query list** consists of words or phrases in some natural language (in almost all cases it is English), linked by Boolean operators. The answer comes as a **ranking list** of documents matching the query, ranked (ordered) so that the most relevant documents appear on the top of it.

Usually here a problem arises, and namely that the list is too long and contains many pages that are not relevant to the query, but rather concern many different topics. Then user must sift through hundreds or thousands of documents to find the few ones that he has been looking for. But, as human patience is limited, usually users after taking a look at the few top documents give over browsing ranking list (quite often not finding the information that they were looking for). According to another survey made by the Search Engine Watch, about 75% of Internet users report searching to be "quite frustrating", and 33% of them finds it "very frustrating". This fact has even been given a name – "**search rage**". The same survey shows that on average users "search rage" if they don't find information that they were looking for in 12 minutes [SearchEngineWatch].

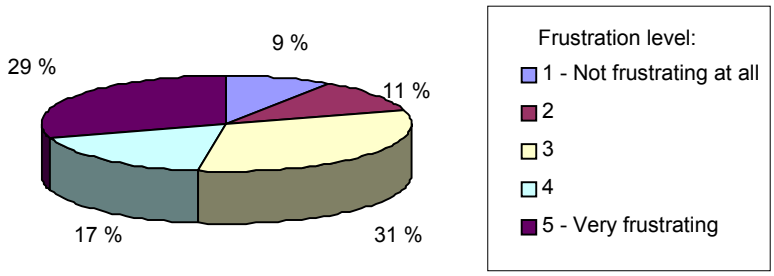

Fig. 1-1-1 Distribution of frustration level of Web search engine users. Values do not sum up exactly to 100% because of rounding [SearchEngineWatch].

As an attempt to improve the quality of search a new paradigm, called **search results clustering**, was formulated. It does not focus on the process of *searching* for the results (it may be, and usually is done completely independent of the searching algorithm), but rather on their *presentation*, so one may say that it is a more "user-centered" approach. Instead of showing the user a long list of results itself, it is split in groups related to sub-topics that occur in the set of documents returned as a result of the query. Then these groups, with a descriptions that enable users to determine their content, are presented. Groups may eventually contain also smaller, more specific groups which may contain further groups etc. – this is called **hierarchical clustering**. There exist two main benefits of clustering:

- the number of groups is much smaller than the number of documents contained in them, so it is easier and faster to browse the results. This fact has been put into the

marketing slogan of the Vivisimo clustering engine: *"No more wasted time scrolling and backtracking through long one-dimensional lists"*.

- the query may concern many different topics (not only those which user had in mind when typing it into the search engine). Clustering reveals these topics as separate groups, so that the user may choose only ones that are interesting for him

Vivisimo claims that *"categorized search results increase the probability of finding key information by an estimated 44%"* [Vivisimo]. Clustering is a natural technique when dealing with a big amount of information – a example of it can be a library catalogue (imagine looking for a book in a library without it, just using a list of items!). Algorithms performing clustering are well known for a long time in computing science, statistics and economy, where they are used for finding groups in *numerical* data [Everitt, 80]. They may be used also to perform a clustering of documents in textual databases [Salton, 89]. Example of such algorithm is **Agglomerative Hierarchical Clustering** (AHC), which generates a hierarchical output.

First really successful search results clustering service was Grouper, developed at the University of Washington [Zamir and Etzioni, 98], [Zamir and Etzioni, 99], which used a brand new algorithm called **Suffix Tree Clustering** (STC). Grouper was a pure research project, and unfortunately is no longer active. Currently the most popular clustering engine is Vivisimo [Vivisimo], which is capable of performing hierarchical clustering. It is a commercial service whose authors do not reveal secrets of used algorithms.

First clustering engine developed in Poland was Carrot, written as a part of master thesis at Poznań University of Technology [Carrot], [Weiss, 01]. Carrot also utilized STC algorithm, trying to apply it to documents written in both English and Polish languages. It was the first clustering service designed for a language different than English. As a continuation of Carrot a new system, called Carrot^2 [Carrot2] was developed. Main purpose of this system was to enable easy performance of experiments comparing different clustering algorithms [Weiss 02a]. This work is also a part of the Carrot^2 system, and focuses on application of Agglomerative Hierarchical Clustering to this problem, as this system doesn't contain implementation of any hierarchical algorithm.

1.2. THE SCOPE AND GOAL OF THE THESIS

This thesis raises subjects associated with domain of computing science called **Information Retrieval**, including problems of processing and searching of text databases [Salton, 89]. This work is particularly strongly connected with another quite new domain partly covering Information4 Retrieval, called **Web Mining,** which concerns similar problems towards Internet (which one also may consider as a large, distributed database). Web Mining raises completely new problems, associated with distributed and chaotic structure of Internet, rapid changes of its content, constraints on time of processing and unknown profile of end user, who may be a ordinary person without any knowledge of how to use the system, which should help him in finding information he needs.

This thesis is supposed to reach two main goals:

- Implementation of Web documents hierarchical clustering algorithm as a part of the Carrot^2 system.

  Choice of existing or design of new clustering algorithm based on so called vector space model (presented in Chapter 3). This algorithm should then be implemented as a module of the Carrot^2 search service. Design and implementation of this module should be compliant with standards of information flow in the Carrot^2 system [Weiss 02a].

- Evaluation of the algorithm's performance on chosen sets of WWW pages (considering also documents in Polish language) and comparison of it and recent approaches to search results clustering like Suffix Tree Clustering or Latent Semantic Indexing.

  In order to achieve this goal, it was necessary to consider measures of quality of algorithm's results, which would consider specific properties of all compared algorithms. Appropriate set of test data, which would allow to study different aspects of these algorithms had also to be created.

1.3. SHORT THESIS SUMMARY

Chapter 2 gives a brief overview of existing answers to the Web search problem, with emphasis placed on search results clustering. Innovative, experimental approaches to presenting search results to the user are also discussed. In the final subchapter we introduce the Carrot^2 clustering engine, which is especially important as this thesis is a part of this system.

In chapter 3 we present the vector space model, which is a classical paradigm in the Information Retrieval domain and underlies the Agglomerative Hierarchical Clustering. Basic terms and assumptions in this model are presented, along with discussion of problems that can be encountered when dealing with natural language.

Chapter 4 presents main documents clustering algorithms, both ones based on the vector spaced model and new ones based on key phrases. A separate subchapter is devoted to a detailed description the AHC algorithm, being a major subject of this thesis. Finally, we give a comprehensive overview of post-processing of the algorithm results which is performed before their presentation to user.

In chapter 5 design and implementation issues of AHC algorithm are discussed, along with a short presentation of the whole Carrot^2 system architecture, which gives a background that may help to understand some of the design solutions.

Main purpose of chapter 6 is an attempt to evaluate the results given by the algorithm. First, we present both the standard quality measures and clustering-specific ones that we have designed. Then, the evaluation experiment and its results are presented in two subchapters.

Chapter 7 ends this thesis presenting once more the main accomplishments and conclusions. Possible future research directions are also listed.

1.4. THESIS LAYOUT

In the first subchapter we have stated that not only the amount of information is important, but also how easy it is to find it. One might say that this sentence also concerns this thesis. So apart from creating the content itself, an effort has also been made to present it to the reader so that she or he may easily find the information that she/he is looking for. Layout of this thesis is based on ones used in many other papers that the author has come across in the past.

Different font styles are used to emphasize special meaning of some parts of thesis. *Italics* is used to place emphasis on parts of text of some special importance. On the other hand, **bold** font appears whenever a new term is defined or introduced.

A standard has also been introduced for the numbering of figures. A decision has been made not to distinguish between figures, tables, equations, etc. All these elements are treated as figures and are numbered according to their order of precedence in chapter in which they appear (e.g. "Figure 4-2-5" would denote fifth figure in chapter 4.2). Respective number and also a short title always appears in the text below each figure. Thereafter figure is referenced in text by its number.

References to other papers and Internet resources are marked using square bracket with a name of the reference. This name consists usually of name of one of the authors of the paper and year of its issue.

2. BASIC CONCEPTS

Main purpose of this fragment of thesis is to give the Reader more detailed knowledge of the Web searching topic. Several types of search services that can be distinguished are discussed here. Most interesting approaches are then discussed in details in subsequent subchapters. First come the indexing services, being still the most often used tool. Techniques that allow them to search and process the vast amount of data found in the WWW are presented. Then we shift to recent interesting approaches to the Internet searching. Most of ideas introduced here are still in experimental phase, but they represent very interesting (and sometimes maybe even unexpected) approaches to Web search problem. We focus our attention especially on search results clustering and Carrot^2 system, which are presented in separate subchapters.

2.1. TYPES OF INTERNET SEARCH ENGINES

Not all Web search engines work in the same manner. Currently several types of services representing different approaches are in use. We will use here a classification based on ones given in [Zamir and Etzioni 99] and [Weiss 02b]. The most important distinction concerns the way of presentation of search results – whether it is oriented toward visualization of attributes of documents or toward visualization of similarities between documents.

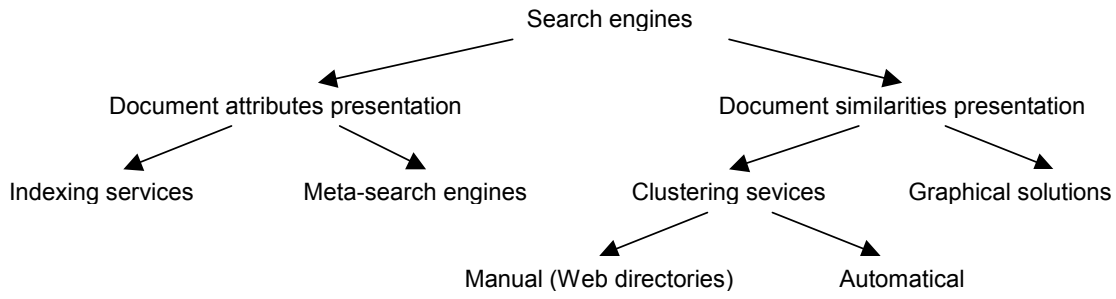

Fig. 2-1-1 Example classification of Internet search services

Historically first type of search engines were so called **indexing services**, which have their own databases containing immense numbers of WWW pages searched in order to find an answer to user's query. Indexing services have always been the most popular tool used for looking for information in the net. The biggest Web search services like Google or Yahoo work this very way. However, as we have already stated, databases of single search engines cover only a small part of Internet information resources. Thus in case of some very specific queries they may give very little or even no results.

In such cases, the simplest solution is to repeat the search using other service, which may give better results. Typing the same query several times and then comparing the result lists is

a time-consuming and boring task, which moreover can be easily automated, and so the **meta-search engines** came to being. Meta-search engines don't have their own databases containing documents, but after receiving the user's query they redirect it to other services, collect the results, and send them back to user. Example services of this type are Dogpile [Dogpile], Mamma [Mamma], Metasearch [Metasearch] or Vivisimo, which in addition performs clustering before presenting results to the user. What is interesting for us is that also Polish meta-search engines, allowing to query Polish search services were built – Emulti [Emulti] and RazDwaTrzy [RazDwaTrzy]. Also client programs that work in a similar fashion do exist (Copernicus or, again for querying Polish Web – Internetowy Poszukiwacz) [Chabiński and Bugajska 03].

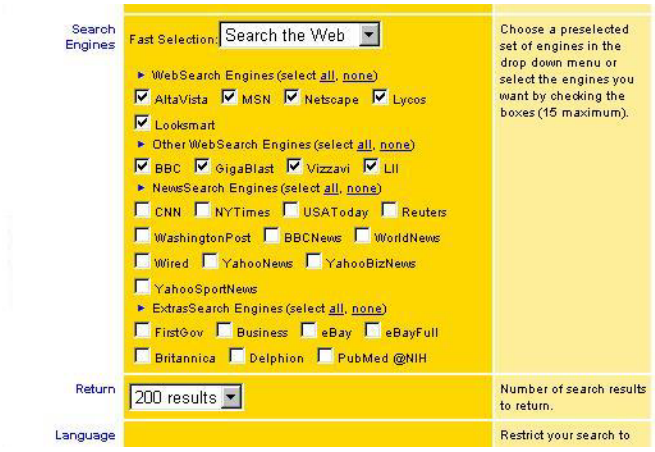

Fig. 2-1-2 Vivisimo's Advanced Search options reveal, which search egines it uses

Due to the fact that different search services may have different formats of query, meta-search engines must translate the user's query before its further broadcasting. There also exist some problems associated with fact that results obtained from different services are disjoint and sorted in different order, so finding a common order for the final ranking list may pose a problem. Meta-search engines can sort the results using their own algorithms, but as they have no additional data, only the collected pages, therefore the results often are poor. A simple test of meta-search engines was performed in [Chabiński and Bugajska 03], and the authors claim that the quality of the results of most of these services (excluding Vivisimo) is rather poor when compared to results given by Google.

Completely different approach are the **Web directories**, working as thematic catalogues of Internet pages, which are *manually* assigned to respective categories. Main directory services are Yahoo, which was the first to employ the idea of directory to search the Web, LookSmart [LookSmart] and Open Directory Project [ODP], which can be viewed as "Open Source" directory – i.e. the documents can be assigned to categories by any Internet user, not only the workers of the company that owns the directory.

This approach is very interesting as it differs widely from classical "query-results list" paradigm, possessing many advantages over it (but unfortunately a lot disadvantages, too). It is also getting more popular (according to [Weiss 02b] a few years ago about a half of Internet users declared using directories when looking for information). It can be also viewed as a

form of clustering, which is done manually, not by the machine. Different, interesting solution was applied in the NorthernLight service [NorthernLight], which is a "half-manual" directory, i.e. categories are predefined by man, but the classification of documents to them is performed automatically.

Service	Editors	Categories	Links	Date
ODP	36,000	360,000	2.6 million	04/01
LookSmart	200	200,000	2.5 million	08/01
Yahoo	> 100	not available	1.5-1.8 million	08/00

Fig. 2-1-3 Comparison of the top Web directories according to [SearchEngineWatch]

Main disadvantages of web directories are:

- slow indexing (due to the fact that it has to be performed manually), for instance the Open Directory Project receives 250 site submissions per hour [SearchEngineWatch]
- relatively small size of databases (example exact numbers are given in Figure 2-1-3), resulting from the former point
- the content of catalogues may be out-of-date, also because of the former point
- predefined directories structure, which may not consider very specific topics

Strong points are:

- hierarchical structure of catalogues, allowing to refine the search, starting from very general topics and moving towards more and more detailed.
- categories are assigned manually, so accuracy is much better than in case of automatic clustering
- there is no need to know any keywords or formulate queries, navigating through directory structure is much more simple and intuitive

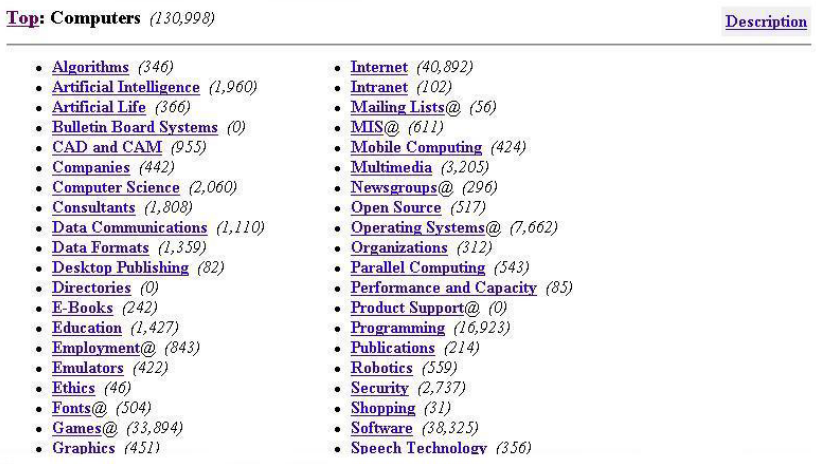

Fig. 2-1-4 Open Directory Project catalogue for category "Computers"

2.2. INDEXING SERVICES

This subchapter concerns typical indexing services using "query-results list" model, and tries to answer how the documents that respond to the user's query are found.

2.2.1. COLLECTING AND STORING THE INFORMATION

In order to find the Web documents relevant to user's query, search services keep indexes of them in their databases. This creates great challenges, as number of these documents may amount to billions and the result has to be found during just few seconds unless we want the user to get impatient. Google has published a short list of example difficulties that they have to overcome [Google], including:

- efficiently searching full index of more than 3 billion documents more than 3,000 times per second at peak traffic times
- crawling and indexing billions of documents, comprising of more than 20 TB of data, in a few days
- running graph algorithms on a graph of 3 billion nodes and 20 billion edges
- keeping index up to the minute by finding and re-indexing almost all web pages within minutes of when they change or they are created

These problems have to be dealt with using distributed and parallel processing techniques as no machine alone has enough processing power. Google takes a very interesting approach here – instead of using powerful workstations, they have employed cluster of over 10,000 cheap PCs to search their databases. Judging simply by the time needed to answer the query (which seems to be most important user's criterion), this solution works perfectly.

Another serious problems for search engine designers are filling the services databases with all documents from the Web and updating them in order to reflect its current state. These problems were put by Selberg [Selberg 99] into questions: *How can all Web pages be located?* (the **Web Discovery Problem**) and: *How can the index be kept up to date with the Web?* (the **Coherence Problem**).

It turns out that for the search services the only way to keep a moderately complete and consistent image of the Web in databases is its constant searching in order to discover new documents or detect changes in ones already contained in database. In most cases one unfortunately cannot rely on any external notifications about changes that have occurred. On purpose to perform Internet searching, special programs called Web **robots** or **spiders** are used, whose task can be illustrated as traversing a graph where nodes represent the documents and edges – hyperlinks between them. When a change is detected, appropriate document is updated, inserted or deleted. Considering size of the Internet, it may take a lot of time to find new pages, and despite the ongoing efforts, search services cannot keep up with abrupt and chaotic WWW growth, covering still smaller and smaller area of it [SearchEngineWatch]. They are only able to capture some snapshot of current state of the Web.

Particularly irritating from the user's point of view is so called "**dead links**" problem, which arises when database of the search engine contains references to documents that don't exist anymore. Research from about few years ago shows that on average several percent of links returned by search services are "dead" [Notess, 99].

2.2.2. MATCHING THE USER'S QUERY

First stage of the searching process is interpretation of user's query. As it was already mentioned in the introduction, it is usually a list of words (or phrases), split by Boolean operators. For instance a query asking about all documents about Oxford expect those containing anything about Cambridge would look like "Oxford AND NOT Cambridge".

This way of formulating queries may pose a lot of problems, which are listed here:

- it forces the users to express them in some artificial language, and the process of "translation" may need to distortions so that query typed into the search engine may mean something different that the one that user had originally in mind.
- user queries are usually too short (in most cases they consist of 1-3 words). Such queries lack context and are ambiguous. Consider for example user who wants to go skiing to Switzerland and types into his favorite search engine a query *"Switzerland"*. But he will be flooded also with lots of information about Swiss watches and chocolate, not only about skiing in the Swiss Alps...
- query list is constructed using Boolean operators, which aren't well known to most of people.
- user may not know exactly what he is looking for (this is also possible, and search engines should be designed with a fact that people don't always know precise questions kept in mind).

As a result of the search, a list of Web pages (depicted on Figure 2-1-3-1) containing given combination of key words from the query is returned (The topic of finding documents relevant to the query is discussed in Chapter 3). Because usually this list is so long that it is impossible to look it through exhaustively, it has to be ordered so that the documents most relevant to the query occupy the first positions and user can focus just on the top of the list.

Fig. 2-2-2-1 Google page showing ranking list of results of a query "king"

Original way to sort the documents was comparing how often do the key words appear in the different pages on the ranking list. Quality of results of this approach unfortunately often is poor, because number of occurrences of certain words isn't the only factor of importance of the document. Moreover, it causes a room for certain abuses to exist, as adding respective words to the page's content, its author may manipulate its position in the ranking.

In order to solve this problem, it was necessary to use information that is provided by structure of mutual references between Web pages. As every experienced Internet (not necessarily – this happens also with books, papers, etc.) user knows, in each domain there are services, which are so called "authorities", i.e. many other pages refer to these services. Moreover, this approach – called **connectivity analysis** may be easy performed automatically. It has become a basis of methods which apart from estimating relevance of document to the query also take into account its "authority". An example of such algorithm is Page Rank [Page et al., 98] used in Google, which was one of reasons why this service achieved its great success.

Apart from ranking list's ordering, a separate problem of creating its content arises. Of course, list cannot consist of whole documents, but giving user just their addresses without any details what they contain wouldn't be a satisfactory solution. So the ranking lists besides links to the documents contains their titles and short, automatically created digests called **snippets**. Snippets should be as short as possible, but on the other hand give the user some information about the page's content so that he can judge if it is of his interest without reading the whole page. Usually snippets consist of sentences that contain keywords, allowing to see the context in which they appear in the document (which, as we have stated before can tell us a lot about exact topic of this document). This also means that for different queries snippets created for the same document may be different.

2.3. AUTOMATIC SEARCH RESULTS CLUSTERING

This approach is based on the **cluster hypothesis**, formulated in [VanRijsbergen 79]. This hypothesis states that similar documents should be relevant to the same queries. So clustering of search results should form from the ranking lists groups of similar documents. Example of documents clustering is shown on Figure 2-3-1:

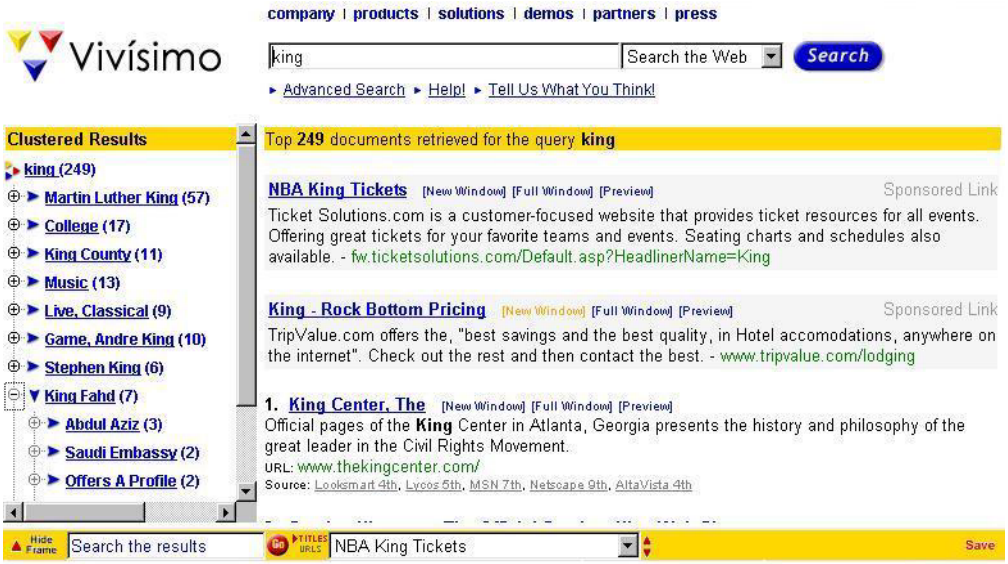

Fig. 2-3-1 Vivisimo page showing clustered results of a query "king"

Automatic document classification is a well-known for a long time approach in the Information Retrieval domain [Salton 89], [Maarek et al. 91], where it was applied to cluster large text databases. This kind of clustering is much easier to perform as these databases aren't often updated as it is in the case of Internet, and the clustering can be performed periodically. This fact also allows to cluster the entire document collection, not only on the query results, which results in creating of hierarchy of the whole database – just like in human-made Web directories. This way of document clustering may be referred to as "**persistent clustering**", contrary to the term "**ephemeral clustering**" which can be used for search results clustering [Maarek et al. 00].

However, the Web is too dynamic environment to use this method, because there aren't any clustering algorithms that allow fast, incremental updating of results which would be required here. Considering also the size of databases of search services, performing a clustering algorithm on their contents would also be a serious problem. Due to this fact, only search results are clustered. It allows also achieving better quality that in case, when all result documents would be assigned to hierarchy created on the basis of whole database (because documents that are not present in the result set have no influence on the way that it is split into topics [Zamir and Etzioni, 99]).

First, pioneer attempts to address the problem of Web search results clustering date to half of the 90-ties. Most important of these systems was Scatter-Gather, described in [Cutting et al., 92]. These approaches were based on grouping techniques from domain of statistics, used

to cluster rather numerical than textual data. As the Web search results clustering must meet some specific conditions (listed further), performance of these "classical" algorithms was quite poor in this area [Zamir and Etzioni, 98], [Zamir and Etzioni, 99]. Breakthrough in this area was the Grouper system, whose author's have succeeded in creating an algorithm that clusters documents in linear time in the size of their set, and performs well when applied to textual data. They have also formulated generic **key requirements for Web document clustering methods** [Zamir and Etzioni 98], [Zamir and Etzioni 99]:

- Relevance

  The method ought to cluster documents relevant to the user's query separately from irrelevant ones.

- Browsable summaries

  The method should provide concise and accurate descriptions of the clusters so that sifting through ranking lists wouldn't be replaced by sifting through clusters. This is a very important issue for document clustering, and also authors of Vivisimo claim that "*rather than form clusters and then figure out how to describe them, we only form describable clusters in the first place*" [Vivisimo].

- Cluster overlap

  Since documents may have multiple topics, the method should allow a document to be assigned to more than one cluster.

- Snippet-tolerance

  The method should produce high-quality clusters even when it has only access to the snippets from the ranking list, not to the whole, original documents as downloading them from the Web would take too much time.

- Speed

  The method should be able to cluster several hundreds of documents (an order of magnitude more than the user might sift through in a ranked list presentation) in a time that would be accepted by the user.

- Incrementality

  This is also some kind of speed requirement which says that in order to save time, the method should start processing each snippet as soon as it is received from the search engine.

2.4. EXPERIMENTAL SOLUTIONS

As we have already stated, many problems with poor quality of search results arise because users have to express their needs as queries in some artificial language. This observation has led to construction of search engines that allow queries in natural language – AskJeeves

[AskJeeves] or AnswerBus [AnswerBus]. This approach is very rare as automatic analysis of natural language is difficult to perform. Another problem is that Internet users speak many different languages (only AnswerBus allows queries in other language than English – namely in French, Spanish, German, Italian and Portuguese), and moreover they tend to make a lot of grammatical and orthographic mistakes, making their queries even harder to understand for the service.

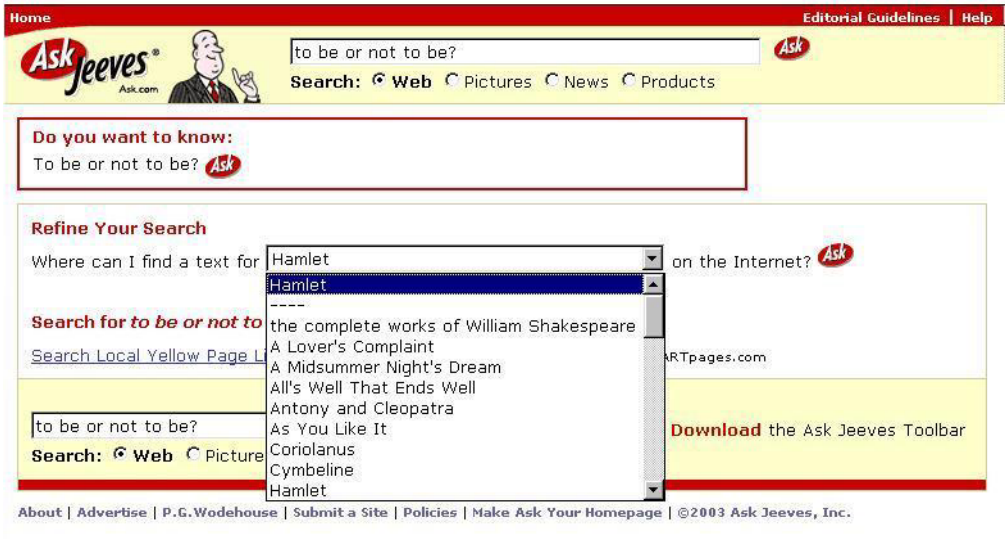

Fig. 2-4-1 AskJeeves page showing results of a query "to be or no to be?"

Remaining services discussed here focus on presentation of similarity between documents returned as query results. It turns out that recently a very popular approach to this problem is graphical visualization of results [Weiss 02c]. One of the most interesting services applying it is Kartoo [Kartoo], which has very nice graphical interface that however may be too "abstract" for user accustomed to typical search engines. Idea used here is based on visualization of documents as nodes of a graph, with relations between them represented by edges (see Figure 2-4-2). These edges are labeled with sets of keywords (resembling somehow labels of groups in document clustering) that are shared by related documents. Also importance of keywords and documents is visualized.

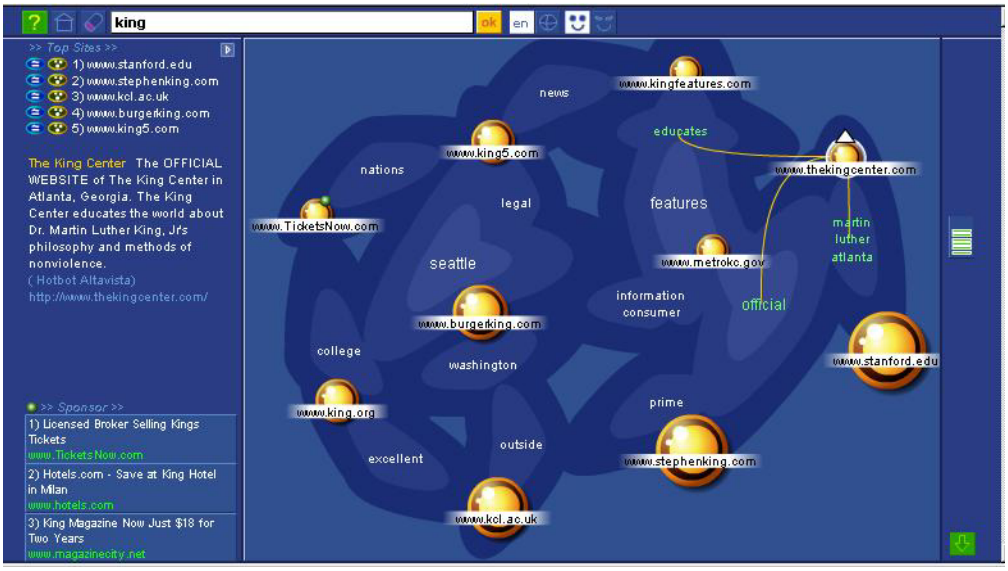

Fig. 2-4-2. Kartoo page showing results of a query "king"

Yet another service of this type is Map.net [MapNet], which – as its name suggests – creates a "map" of the results. This search engine uses Open Directory Project catalogue, dividing Internet into categories present there, and plots result documents on map putting them in areas corresponding to their topic. Size and "geographic position" of areas on the map depend on number of documents in respective categories, and mutual relations between them.

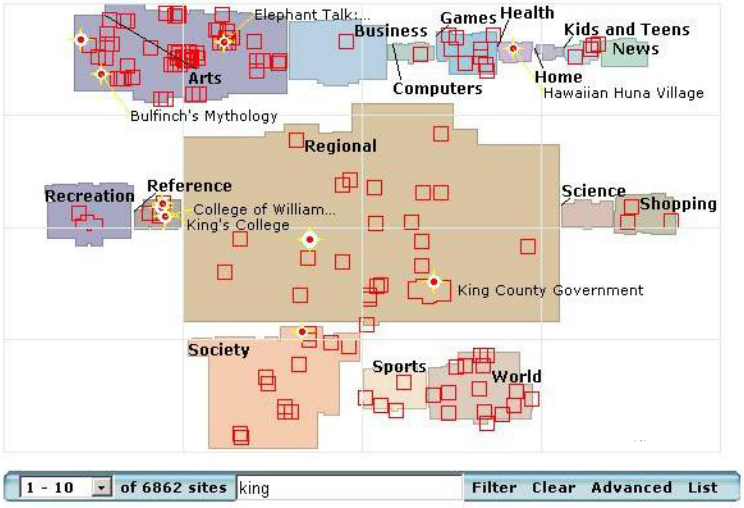

Fig. 2-4-3 Map.Net page showing map of results of a query "king"

Both these systems are very interesting and the way they present the results is really more and user-friendly when compared to the ranking list. But they are still in experimental phase and should be considered rather as fads, not practical tools. And it seems that still nothing is going to threaten (anyway well-earned) Google's market position in the next few years.

2.5. THE CARROT^2 SYSTEM

The Carrot^2 system originated from Carrot clustering meta-search engine, which was a result of work described in [Weiss 01]. Main goals of Carrot included evaluation of applicability of Web document clustering to Polish language and – what's more important from this thesis point of view – creating an *open-source* clustering system, which could form a basis for further work. This goal was successfully realized and system is further developed under the name of Carrot^2. Main improvements consider the system's architecture in order to increase its flexibility [Weiss 02]. Thanks to these changes, easy adding or changing of components of the system is possible, which is necessary in order to easy perform experiments with different algorithms (which are illustrated below on Figure 2-5-1 and Figure 2-5-2 and Figure 2-5-3)

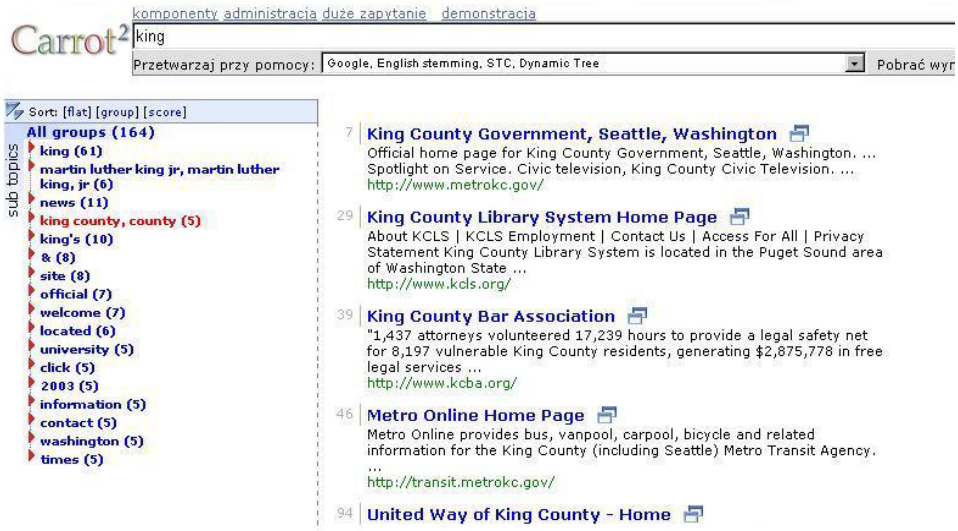

Fig. 2-5-1 Carrot2 page showing clustered results of a query "king" (Suffix Tree Clustering algorithm used)

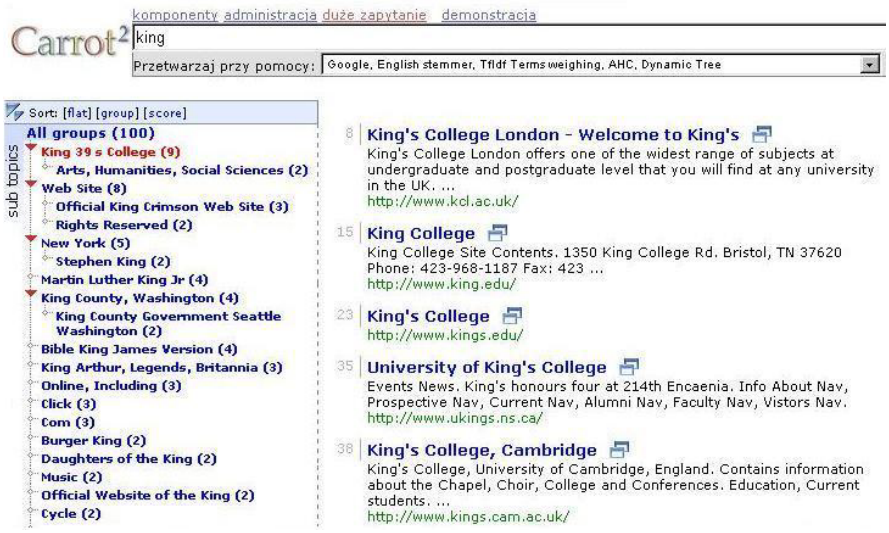

Fig. 2-5-2 Carrot2 page showing clustered results of a query "king" (Agglomerative Hierarchical Clustering algorithm used)

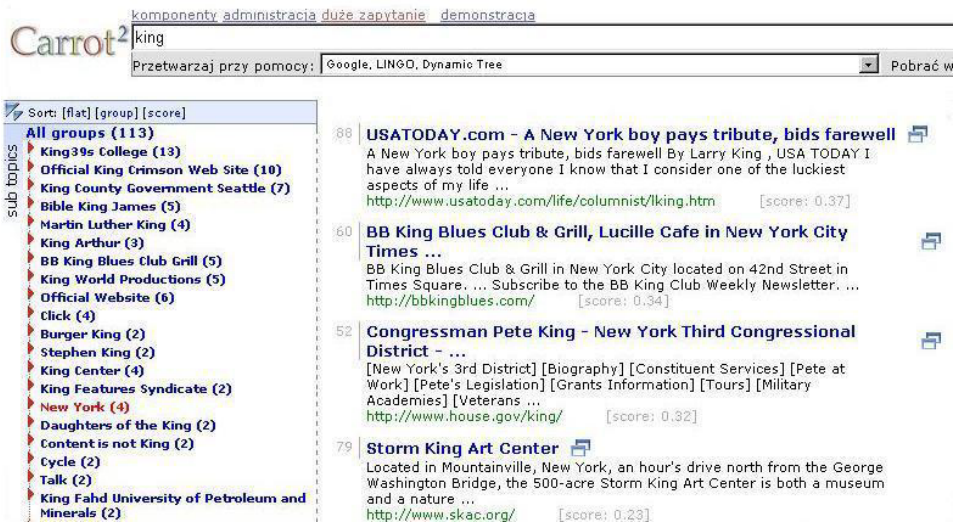

Fig. 2-5-3 Carrot2 page showing clustered results of a query "king" (LINGO algorithm used)

Main problems concerning further development of Carrot^2 are – although it may seem unexpected – legal issues. As it is a meta-search engine, it doesn't have a indexed pages database of its own. Instead, Carrot^2 performs clustering on search results collected from other search engines. As it turned out, this kind of "parasitic" activity is considered illegal by terms of use of commercial services. Fortunately, a contact has been established with authors of new, experimental indexing service Egothor [Egothor], who have declared that Carrot may make a use of results returned by it, and now it finally may be possible for Carrot to have a legal "background" search engine.

3. THE VECTOR SPACE MODEL IN INFORMATION RETRIEVAL

In this chapter a comprehensive overview of the vector space model, being a classical approach applied to processing databases containing text documents is given. First, we present the basic concepts in this model, then documents representation and how documents corresponding to a query, or similar to other one may be found. This model has some severe drawbacks, resulting from its main assumption – reducing texts written in natural language, which is very flexible to strict, mathematical representation. These problems, along with their possible solutions are also discussed.

3.1. BASIC CONCEPTS

3.1.1. DOCUMENTS REPRESENTATION

The vector space model is based on linear algebra and treats documents and queries as vectors of numbers, containing values corresponding to occurrence of words (called here **terms**) in respective documents [Salton et al. 75]. Let t be size of the terms set, and n – size of the documents set. Then both the query Q and all documents D_i, $i = 1..n$ may be represented as t-dimensional vectors:

$$D_i = [a_{i1}, a_{i2}, ..., a_{it}] \qquad Q = [a_{q1}, a_{q2}, ..., a_{qt}]$$

Fig. 3-1-1-1: Document and query representation in the vector space model

where coefficients a_{ik} and a_{qk} represent the values of term k in document D_i or query Q, respectively [Salton 89]. Thus both documents and terms form a **term-document matrix** $A(n \times t)$. Rows of this matrix represent documents, and columns – so called **term vectors**. Let us assume that position a_{ik} is set equal to 1, when term k appears in document i, and to 0 when it doesn't appear in it (next subchapter is devoted to the topic of calculating of these values). Then for example documents collection corresponding to a query *"king"* we could create a corresponding term-document matrix (see Figure 3-1-1 below):

Documents set:
D_1: The King University College
D_2: King College Site Contents
D_3: University of King College
D_4: King County Bar Association
D_5: King County Government Seattle Washington
D_6: Martin Luther King

Terms set:
The, King, University, College, Site, Contents, of, County, Bar, Association, Government, Seattle, Washington, Martin, Luther

Term-document matrix:

1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	1	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	1	1	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	1	1	1	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1	1

Fig. 3-1-1-2: Term-document matrix for an example document collection

It is easy to see that documents corresponding to similar topics should have similar values in the same positions of their vectors.

3.1.2. STRONG AND WEAK POINTS OF THE MODEL

Main advantages of this paradigm are:

- Linear algebra as the basis of the model

After transforming documents to vectors we may easily perform mathematical operations on them, using methods of linear algebra, which is a well known and studied sub-domain of mathematics.

- Simple, efficient data structures may be used to store data

Representation of documents in the vector space model is very simple, and we may store them efficiently using arrays. However, often these arrays are **sparse**, i.e. most of contained values are equal to 0 (especially when considering large numbers of documents containing many terms). Special methods for storing and processing such vectors may be used in order to decrease the amount of time and memory needed [Duff et al. 02]. However, they are beyond the scope of this thesis and we will not discussed these issues.

Basic weakness of this model is

- Lack of information contained in the *structure* of documents

In basic vector space model, only *occurrence* of words in documents is important (i.e. it is treated as a *set* of terms), and their *order* isn't considered. It is the main reason why this approach is often criticized ([Zamir and Etzioni 98], [Weiss 01]), as the information about the proximity between words (their context in sentence) is not utilized. Consider for example two documents: one containing a phrase *"White House"*, which has a very specific meaning, and another containing a sentence *"A white car was parked near the house"* Treating documents simply as sets of terms we only know that words *"white"* and *"house"* occur in both documents, although their context there is completely different.

However, this problem can be easily overcome – one can supplement this model, using also *phrases* in addition to terms in document vectors, as described in [Maarek et al. 91] and [Maarek et al. 00]. This approach will be discussed in the last subchapter of this part of the thesis.

3.2. ASSIGNING WEIGHTS TO TERMS

3.2.1. METHODS OF TERMS WEIGTHING

As we have mentioned in the former subchapter, values at individual positions of document vectors correspond to occurrence of terms in these documents. These values are called **weights**, as they describe how *important* is this term in context of the containing document (the same term may have *different* weights in different documents). Process of calculating weights of terms is called **terms weigthing**. There are several main methods used to assign weights to terms.

The simplest method is **boolean terms weigthing**, which – as its name suggests – sets weights to 0 or 1 depending on the presence of term in document. This method was used to calculate the term-document matrix in example shown on Figure 3-1-1-2. Using this method causes loss of valuable information, as it differentiates only between two cases: *presence* an *absence* of term in document, and exact number of occurrences of word may indicate is's importance in documents [Salton 89].

Method utilizing knowledge of exact number of term occurrences in documents is called **tf terms weigthing** (tf stands for term frequency). It assigns to each position of document vector value equal to number of corresponding term's occurrences in this document (so when it is used, same term may have different weight in different documents). This weight is linear in the number of occurrences, and it may rise too fast (i.e. word appearing 5 times as often doesn't necessarily have to be 5 times as relevant), so some type of "smoothing" may be used here (for instance square root – [Salton and Buckley 87]).

Another significant issue is frequency of appearance of term in the *whole* collection of documents, not only in a single one. This is especially important for clustering purposes, where we have to *split* documents into some subgroups, so words occurring only in a small

subset of the whole collection (suggesting that it may form a cluster) should be rewarded. Number of documents that contain term t_i is called the **document frequency** of term t_i and denoted by df_i [Salton 89]. Using this measure, one may calculate **inverse document frequency** of term t_i (idf_i) in the following manner:

$$idf_i = \log_n \frac{n}{df_i}, \; n - number\ of\ all\ documents$$

Fig. 3-2-1-1: Formula for inverse document frequency [Salton 89].

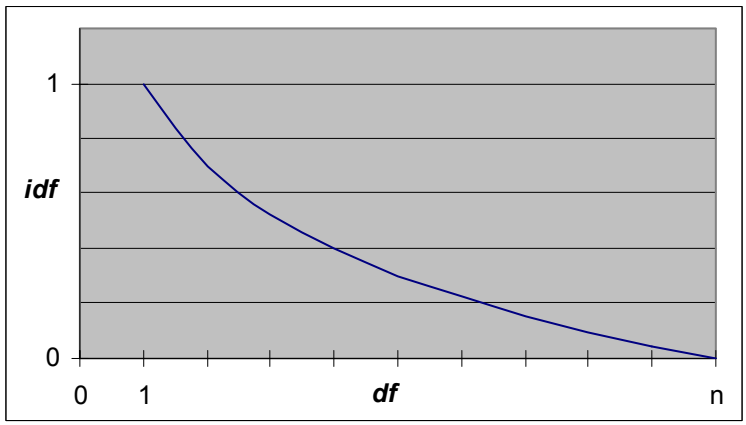

Fig. 3-2-1-2: Inverse document frequency as a function of document frequency

A classical terms weigthing method that takes into account both *term* and *document* frequencies is called **tf-idf terms weighting**, and is probably the most popular approach in information retrieval systems. Term weight in this method is calculated as a product of its term and inverse document frequencies, hence its name.

$$w_{ij} = tf_{ij} \cdot \log_n \frac{n}{df_i}, \; n - number\ of\ all\ documents$$

Fig. 3-2-1-3: Formula for tf-idf terms weighting [Salton 89]

We have discovered that unfortunately using formula for inverse frequency presented in Figure 3-2-1-1 causes weight of term to drop too rapidly with increase of its document frequency (words that occur in many documents are too strongly penalized). As a result, it may be difficult to form larger clusters of documents (because words common for these documents have a very small weight). Therefore we propose another formula for calculating inverse document frequency, whose value decreases *linearly* in the number of documents in which the term occurs (sample results of different terms weighting methods are compared in Chapter 6-3-6):

$$idfi = 1 - \frac{df_i - 1}{n - 1} = \frac{n - df_i}{n - 1}$$

Fig. 3-2-1-4: Formula for inverse document frequency, n – number of all documents

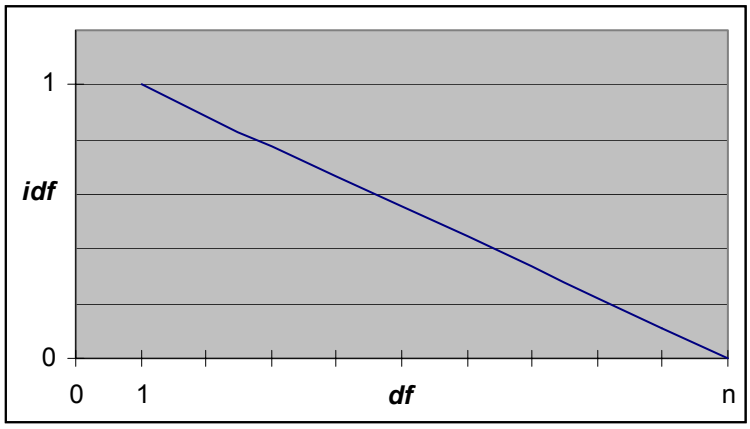

Fig. 3-2-1-5: Inverse document frequency (using formula from Fig. 3-2-1-4) as a function of document frequency

3.2.2. TERMS WEIGHTING ISSUES SPECIFIC FOR SEARCH RESULTS CLUSTERING

According to Zamir and Etzioni's key requirements for Web documents clustering methods (presented in Chapter 2-3), the only data that a search results clustering system may use, are *snippets* and *document titles*. Although search engines return usually snippets of good quality (i.e. ones being a good digest of document, enabling to deduce its topic), this restriction still means that we have to perform clustering using incomplete data. Therefore during the process of terms weighting we don't know how *really* often do the terms occur in documents and may, for instance, assign a low weight to a term that appears many times in a document (but the snippet contains only a few occurrences of it).

So occurrences of terms in snippets may seem random and that's why it is sometimes suggested to calculate the *idf* factor on some considerably large document set instead on search results [Weiss 01]. However, we cannot agree with this approach, as the distribution of terms in the query result set (which is usually associated just with a few topics) may differ significantly from their distribution in a large text set, containing documents from many domains. For instance, a word *"snippet"* would have a very high document frequency in a collection of documents responding to a query *"search results clustering"*, but in general it is rather a rarely occurring term. Therefore, we have decided to calculate terms weights on the basis of the input snippets collection.

During work at this thesis we have used also other ideas that may partially improve search clustering results:

- Increasing weights of terms that appear in titles of documents

  According to assumptions of this method, if a term appears not only in snippet but also in *title* of an input document, its weight in *this* document should be scaled by some factor ([Osiński 03], where factor value of 2.5 is proposed). This is due to the fact that titles, given to documents by their authors, should be treated as their best digests. We have also used this method, but haven't observed any considerable, unambiguous improvement.

- Removing terms that appear only in one document

  As matrices created during terms weighting are sparse, some authors postulate that **dimensionality reduction** should be performed [Karypis and Han 00]. Main goal of this process is removing from the matrix rows corresponding to the *least important* terms. It is usually performed by applying complicated matrix transformations. We propose here a simple approach, consisting in removing from term-document matrix rows corresponding to terms (and phrases – when using them) that occur only in one document (i.e. all terms t_i such that $df_i = 1$). Such words don't aid in forming clusters, as they cannot be shared by multiple documents, and may be treated as a noise. This method was introduced also for performance reasons (about two thirds of terms occur only in one document due to the mentioned earlier fact that term vectors are sparse), allowing to decrease size of term-document matrix and therefore to achieve a considerable speed-up in its computations.

- Removing term that is equal to the user's query

  Forming a group from documents that share the query as a common term would be rather absurd. Moreover, this group would probably contain almost or even all result documents as most of results contain term that was given in the query. When using phrases, this condition means that phrase equal to the query (which may also be a phrase itself) and its subphrases are not added to the terms vector.

3.3. DOCUMENT SIMILARITY MEASURES

Using vector representation, it is easy to find documents relevant to the query or similar to each other by calculating a similarity value of (or distance) between vectors either representing document and query, or representing a pair of documents. Calculating similarity of two t-dimensional vectors is a well-known mathematical problem, and can be done in many ways. Example similarity measures between two vectors are given in [Salton 89]:

Similarity measure sim(X, Y)	Evaluation for binary term vectors	Evaluation for weighed term vectors
Inner product	$X \cap Y$	$\sum_{i=1}^{t} x_i \cdot y_i$
Dice coefficient	$2\frac{\lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert}$	$\frac{2\sum_{i=1}^{t} x_i y_i}{\sum_{i=1}^{t} x_i^2 + \sum_{i=1}^{t} y_i^2}$
Cosine coefficient	$\frac{\lvert X \cap Y \rvert}{\lvert X \rvert^{1/2} \cdot \lvert Y \rvert^{1/2}}$	$\frac{\sum_{i=1}^{t} x_i y_i}{\sqrt{\sum_{i=1}^{t} x_i^2 \cdot \sum_{i=1}^{t} y_i^2}}$
Jaccard coefficient	$\frac{\lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert - \lvert X \cap Y \rvert}$	$\frac{\sum_{i=1}^{t} x_i y_i}{\sum_{i=1}^{t} x_i^2 + \sum_{i=1}^{t} y_i^2 - \sum_{i=1}^{t} x_i y_i}$

Fig. 3-3-1 Example similarity measures between two vectors

Unfortunately, choice of a particular similarity measure for certain application lacks theoretical justification and is rather arbitrary [Salton 89]. The most commonly used measure is the cosine coefficient [Maarek et al. 00], which is equal to the cosine of *t*-dimensional angle between two compared vectors.

Very useful property of a similarity measure is **normalization** of its values. Normalized similarity coefficients have values varying always between 0 and 1 when the vector elements are nonnegative (which is always the case for term vectors). Knowledge of range of possible values of a coefficient enables us to compare its values for different pairs of vectors. Typical measures that have this property are Dice, Jaccard and cosine coefficient (see Figure 3-3-1), whereas values of the inner product are practically unbounded [Salton 89] and have to be normalized later after their calculation. Moreover, inner product takes into account only how many *common* terms occur in two compared vectors, completely ignoring number of differences between them. Thus two exemplary pairs of sentences: ("King's College", "King's College") and ("The University of King's College", "King's College London") will have the same similarity value according to this measure, as they both share 2 same terms.

Another question concerning similarity measures is an association between *similarity* and *distance* of vectors. Measures presented in Figure 3-3-1 produce results that show how two vectors are similar to each other, but there are also other coefficients (for instance the widely-used Euclidean measure shown on Figure 3-3-2), whose results indicate distance of vectors. In order to compare results of those two approaches, some way of conversion of their values must be proposed.

$$dist(X,Y) = \sqrt{\sum_{i=1}^{t} (x_i - y_i)^2}$$

Fig. 3-3-2 Euclidean distance between two vectors

As similarity values are usually normalized between 0 and 1, it is a common practice to define the distance between term vectors as 1 minus their similarity [Maarek et al. 00]. On the other hand, for converting distances to similarities a formula (presented on Figure 3-3-3) below is used. This formula is taken from [Everitt 80], where this topic is discussed in more details.

$$sim(X,Y) = \frac{1}{1 + dist(X,Y)},\ dist(X,Y) - distance\ between\ two\ documents$$

Fig. 3-3-3 A formula used to convert from distance to similarity measure [Everitt 80]

3.4. DEALING WITH NATURAL LANGUAGE

This subchapter is dedicated to presentation of difficulties that may arise during conversion of written texts to vector representation because of complexity of natural languages. This subchapter is actually the only parts of the thesis, where described algorithms are dependent on the language of clustered documents. It means that during development of a multilingual clustering system, a lot of work may be needed in this area.

3.4.1. STOPWORDS

Many words occurring in text contain no information about its *topic*. These are so called **function words** [Salton 89], which perform some necessary function in sentence structure (e.g. link two words – *"and"*, *"but"*, are used to form different tenses – *"will"*, *"have"*, etc.), but have no meaning themselves. These words shouldn't be considered during processing of the text, as they are not only meaningless, but moreover occur very frequent in normal sentences. So their usage could lead to forming strong, but nonsense clusters – for instance from documents that share a word *"of"*, *"and"* or *"the"*.

This problem is solved by excluding these terms from terms weighting. A list of function word, called **stop-list** is used and words from this list (called **stop-words**) are treated as meaningless. Example stop-list for English language (it is obvious that stop-lists have to be constructed separately for each language) used in the Carrot^2 system is shown on Figure 3-4-1-1. This list is of course constructed before clustering and remains the same for different queries. It may be either constructed manually by linguistic experts or automatically by analyzing frequencies of different words in a **text corpus** (i.e. some set of texts that is big enough to allow statistical processing of it and to recognize it as representative for given language) and choosing as stop-words terms that are most common [Weiss 01]. In the second case, the list also usually needs some manual correction.

Putting to stop-list only words performing some grammatical function may not be enough. Usually Web documents contain some words associated with Internet (*"link"*, *"www"*, *"url"*), which perform some other functions in text of the document (as navigation between Web sites), but contain no information. These terms should be also added to the stop-list.

a	becoming	even	his	namely	same	therefore	whenever
about	been	ever	how	neither	see	therein	where
above	before	every	however	never	seem	thereupon	whereafter
across	beforehand	everyone	hundred	nevertheless	seemed	these	whereas
after	behind	everything	i	next	seeming	they	whereby
afterwards	being	everywhere	ie	nine	seems	thick	wherein
again	below	except	if	no	serious	thin	whereupon
against	beside	few	in	nobody	several	third	wherever
all	besides	fifteen	inc	none	she	this	whether
almost	between	fill	indeed	noone	should	those	which
alone	beyond	find	interest	nor	show	though	while
along	both	first	into	not	side	three	whither
already	bottom	five	is	nothing	since	through	who
also	but	for	it	now	sincere	throughout	whoever
although	by	former	its	nowhere	six	thru	whole
always	call	formerly	itself	of	sixty	thus	whom
am	can	forty	keep	off	so	to	whose
among	cannot	found	last	often	some	together	why
amongst	cant	four	latter	on	somehow	too	will
amoungst	co	from	latterly	once	someone	top	with
amount	con	front	least	one	something	toward	within
an	could	full	less	only	sometime	towards	without
and	couldn	further	ltd	onto	sometimes	twelve	would
another	cry	get	made	or	somewhere	twenty	yet
any	describe	give	many	other	still	two	you
anyhow	detail	go	may	others	such	un	your
anyone	do	had	me	otherwise	system	under	yours
anything	done	has	meanwhile	our	take	until	yourself
anyway	down	have	might	ours	ten	up	yourselves
anywhere	due	he	mill	ourselves	th	upon	home
are	during	hence	mine	out	than	us	page
around	each	her	more	over	that	very	http
as	eg	here	moreover	own	the	via	www
at	eight	hereafter	most	part	their	was	java
away	either	hereby	mostly	per	them	we	link
back	eleven	herein	move	perhaps	themselves	well	url
be	else	hereupon	much	please	then	were	
became	elsewhere	hers	must	put	thence	what	
because	empty	herself	my	rather	there	whatever	
become	enough	him	myself	re	thereafter	when	
becomes	etc	himself	name	s	thereby	whence	

Fig. 3-4-1-1 An example stop-list for English language used in the Carrot system [Weiss 01]

3.4.2. STEMMING

Presented in chapter 3.2 methods of terms weighting treat different sequences of characters as separate positions of term vectors. But it turns out that two quite *different* strings can represent one word. It is a result of inflection, which is present in natural languages and may cause a word to occur in different grammatical forms, depending for instance on its number (*car – cars*) or degree (*slow – slower – slowest*). Also several different words may originate from one, and have a one common meaning (example: *information – inform – informative*). All these words represent a similar sense and should be treated as one position in the terms vector.

A solution to this problem is **stemming**, consisting in reduction of all forms of a word or words to one, common part called **stem**. After performing of stemming, the position of terms

vector correspond to *stems*, not to single words. An example of stem and its different forms is shown on the below Figure 3-4-2-1:

stem	**forms**
ANALY	analysis
	analyses
	analyst
	analyse
	analysing
	analytical

Fig. 3-4-2-1 An example of stemming, taken from [Salton 89]

Issue of stemming is particularly well studied in case of English language. A lot of **stemmers** (i.e. stemming algorithms) have been create for this language. Main reasons of this fact are that, firstly it has quite simple syntax and strict inflection rules, making the process of stemmer development easy, and secondly of course English is the language of computing science. Example stemmers for English language are Lovins stemmer [Lovins 68] and Porter stemmer [Porter 80]. In the Carrot^2 system, the Porter stemming algorithm is used. It is a small piece of code, being just a set of several quite simple rules (steps).

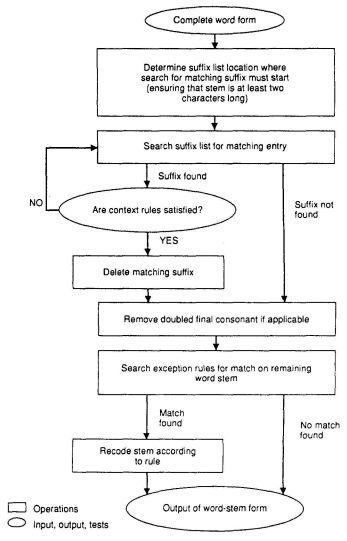

Fig. 3-4-2-2 An example of a stemmer algorithm [Lovins 68]

Since one of most important goals of the Carrot^2 system is clustering of documents in Polish language, we will describe here shortly also stemming for this language. This issue has already been addressed in details in [Weiss 01], and we will mention here only the most important facts.

Polish language is one that has quite complex inflection rules, with a lot of exceptions. Therefore stemmers for it have to be created manually by linguistic experts on the basis of

dictionaries, which makes process of their development expensive, and none is available for free. So during development of the original Carrot system a need arose to create one, and a simple stemmer based on automatic analysis of text corpus was created. This software is used now in the Carrot^2 system for stemming of Polish documents.

3.4.3. SYNONYMS

Apart from the fact that one word may have different forms, also many *different words* that have different stems may have similar or the same meaning (for example – *"cluster"* and *"bunch"*). Synonyms may be used often in written documents by their authors to avoid repetition of the same word, and they are treated as different, not correlated terms. In order to deal with synonyms, a thesaurus has to be used, enabling to check if a word has the same meaning as other ones occurring in the document. All terms containing the same information are usually replaced by one common, artificial identifier which is used instead of many words during the terms weighting phase. This process is called in [Salton 89] **thesaurus transformation**.

Such thesaurus has to be created manually, and its construction takes a lot of resources (especially time), so we couldn't develop our own one. Instead we would have to choose an existing one (which would be available for free). Example of such system is WordNet [WordNet], developed at the Princeton University. It contains most of word in the English language, and also the relations that exist between them, including synonymy (which is only one possibility, information about other relations such as hypernymy representing a hierarchy of words is also available). Unfortunately, due to the lack of time we have made no use of WordNet (anyway, using it would enable us only to detect synonyms in English documents, we don't know about any tool for Polish language). Using synonyms was rather a minor goal as we have observed that they occur quite rarely in documents, and therefore this approach wouldn't improve clustering quality significantly.

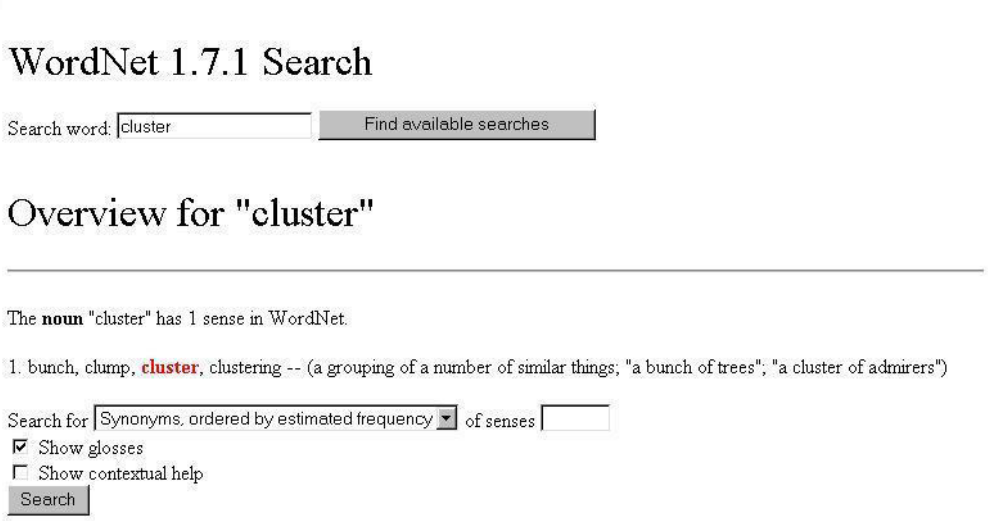

Fig. 3-4-3-1 An example of (online) usage of the WordNet system

3.5. USING PHRASES IN THE VECTOR SPACE MODEL

3.5.1. REASONS TO USE PHRASES

Natural way to represent phrases in the vector space model is to simply treat them as elements of the terms vector so that weights may be assigned to them and then they can be included during vector similarity computations. For example set of sentences presented earlier in Figure 3-1-1-2, valid phrases could be *"King College"* or *"King County"*. As we have mentioned earlier in Chapter 3-1-2, such phrases carry much more information than just simple occurrences of terms (in this case – *"King"*, *"College"* and *"County"*) in document.

Nevertheless there has been a number of opinions that using phrases in the vector space model is a rare approach [Zamir and Etzioni 99] and that it is not likely to achieve significantly better results than using only single words [Salton 89]. There are also some papers showing successful application of phrases [Maarek et al. 91] [Maarek et al. 00]. In our opinion, main benefit of using phrases consists rather in better quality of group labels (see Chapter 4-3-2) than the quality of clustering itself. Phrases are more "expressive", and make the descriptions more *concise* and *accurate*. Consider for example a group of documents about using Agglomerative Hierarchical Clustering algorithm to search results clustering. This group could be described with three phrases *"agglomerative hierarchical clustering"*, *"search results clustering"*, and *"vector space model"*. Without phrases its label would change into something like *"space"*, *"clustering"*, *"model"*, *"hierarchical"*, *"results"*, *"vector"*, *"agglomerative"*, *"search"*, and even if this cluster would contain documents relevant to the description, it would take the user much more effort to guess its topic from the description. In Chapter 6-3-3, we show benefits from phrases use on sample algorithm results.

3.5.2. PHRASES EXTRACTION

In order to treat phrases as elements of the terms vector, they have first to be found in the input documents set. This process is called the **phrases extraction**. Phrases are built from single words, so the only possible way to retrieve them without additional information is to consider all sequences of words in documents (up to some length) and choose from them ones that may represent valid phrases.

Phrases retrieval is performed *separately* for *different* length of sequences from 2 up to some arbitrarily chosen maximum length (we have found 5 to be enough, as phrases longer than 5 words occur in documents very rarely). Each iteration of the process consists of two stages. In the first one, all sequences (called also **n-grams**) of words of given length are found in the documents using algorithm that is presented below on Figure 3-5-2-1. It is a simplified version of method presented in [Maarek et al. 91] and [Smadja 93].

```
// retrieving all sequences of terms of given length from a single given
// document (after stemming terms and removing stop-words, with sentence
// boundaries marked by dots)
for (i = 0; i < (document.length - length); i++)
{
    String[] terms;
    for (j = 0; j < length; j++)
    {
          term = document[i + j];
          if (term == ".") break;
          else terms[j] = term;
    }

    if (j == length)
    {
          // found a sequence, consisting of terms[0] + terms[1] + ...
          // + terms[length-1]
          ....
    }
}
```

Fig. 3-5-2-1 Algorithm used to retrieve possible phrases of given length

Result of this algorithm are all n-grams of words of given length (not crossing sentence boundaries) that are present in the given document, so it has to be repeated for all the documents from the whole collection. It is also easy to see that this algorithm's complexity is linear in number of words in the document. This means that retrieving sequences from all documents is linear in their number (as number of words in one document may be treated as a constant).

Only a small part of sequences produced by this method represent *valid*, *important* phrases, so a second stage is needed – sifting through the candidates in order to find the significant phrases. This is done using only information about their frequencies. For all n-grams of current length, their frequency, its average value and deviation are calculated. Then for each one its **strength** [Smadja 93] is calculated. Value of a strength of a sequence *s* tells us how many standard deviations above the average frequency of all sequences of current length is the frequency of *s*:

$$k(s) = \frac{f(s) - E(f_n)}{\sigma(f_n)}, \quad n - \textit{current length}$$

Fig. 3-5-2-2 Definition of a n-gram strength

The most frequent n-grams (i.e. possessing certain minimal strength) are considered as important and attached to the term vector. Then the process is repeated for the next value of length.

In [Smadja 93] the value suggested for strength is 1, which should filter out the majority of insignificant phrases. Our experiments have shown that using strength value of 1 for a document collection of 100 snippets usually meant that a sequence had to occur at least 3 times in the documents set in order to be classified as important.

3.5.3. POSSIBLE IMPROVEMENTS OF PHRASES EXTRACTION

Methods presented by us in this chapter are very simple and may be further improved. In particular, it does not address the following issues:

- using grammatical information during phrases retrieval

  Most of publications about phrases extraction comes from the lexicography domain and their authors postulate that the statistical significance of a retrieved phrase alone is not enough – the algorithm should also check its grammatical correctness ([Church et al. 91], [Smadja 91] [Smadja 93]). Also [Salton 89] states that *"phrase-formation process controlled only by word co-occurrences and the document frequencies of certain words is not likely to generate a large number of high-quality phrases"*.
  In order to perform grammatical validation, we need for *every* word information about the corresponding part of speech and also knowledge of the syntax rules (which both are again language-dependent). We didn't have such data and have extracted the phrases using only the statistical information. Moreover, it turned out that in fact grammatical information is not necessary for our application of phrases extraction – we are interested just in the mere fact of occurring some words together in different documents as it may point that they are somehow related.

- consideration of the order of terms

  Sometimes phrases that contain the same terms, but in different order have the same meaning. This point is partially connected with the previous one, as importance of order of words depends on the documents language and used grammatical constructions. Using an example from [Weiss 01], in English two phrases: *"John hit Paul"* and *"Paul hit John"* carry completely opposite meaning, while in Polish sentences *"Jan uderzył Pawła"* and *"Pawła uderzył Jan"* mean exactly the same.

- using complete phrases only

  When using our method, if a document contains a phrase longer than two terms, also its subphrases may be additionally extracted and treated as separate ones (for instance, from a document containing a sentence *"King County Seattle Government"*, also phrases *"King County"*, *"County Seattle"*, etc. would be retrieved). It is often postulated that only **complete phrases** should be used, i.e. in case of our example – *"King County Seattle Government"*, in order to avoid redundancy..

- methods of weighting specific for phrases

  Extracted phrases are treated during construction of term-document matrix just like terms, i.e. their weights are calculated on the basis of their occurrences in documents. However, one may try also to make use of phrases length during this process, as usually longer phrases are more important and informative to the user.

4. DOCUMENTS CLUSTERING ALGORITHMS

Main purpose of this chapter is to present to the Reader the most important clustering algorithms, taking especially into consideration Agglomerative Hierarchical Clustering as it is major subject of this thesis. In the first part, a short overview of the clustering problem is given. Several algorithms (including also ones based on the vector space model) are presented in order to show different approaches to this issue. Second subchapter is dedicated completely to the detailed presentation of Agglomerative Hierarchical Clustering. As this method is in reality just a *framework* of algorithm, and may be widely interpreted, we present its main versions and describe differences between them. Finally, we discuss techniques applied by us in order to adapt the AHC algorithm to our application, i.e. Web search results clustering.

4.1. OVERVIEW OF MAIN CLUSTERING ALGORITHMS

Before discussing clustering algorithms, we should present the main idea of clustering itself. A good (although a little too narrow for our purposes) definition is given in [Everitt 80]:

"Given a number of objects or individuals, each of which is described by a set of numerical measures, devise a classification scheme for grouping the objects into a number of classes such that objects within classes are similar in some respect and unlike those from other classes. The number of classes and the characteristics of each class are to be determined."

This definition emphasizes the most important properties of clustering:

- Objects within the formed clusters should be *homogeneous*, while the clusters should be *heterogeneous*. It is necessary, as clusters should help us distinguish between objects.
- Number and attributes of clusters should be *found out* by the algorithm, not given as the input data. So it should not only assign object to groups, but also determine their structure. That's why clustering is sometimes called **unsupervised learning**.

Example of clustering of numerical data, consisting of a set of objects described by two variables (so that they may be interpreted as points in two-dimensional space) is given on below Figure 4-1-1. These points have been classified into five distinct groups.

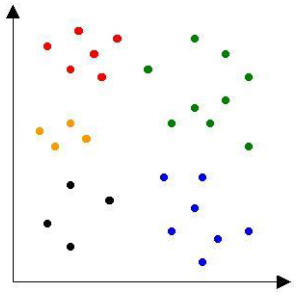

Fig. 4-1-1 Clustering example

However, mentioned earlier clustering definition misses one important point – it supposes that data is described always by *numerical* measures. It doesn't have always to be true, as clustering is used in many different domains and one might for instance wish to find clusters in other kind of data like images or text documents (which is our case). Most of known clustering algorithms were designed for numeric data, so it may seem easier to transform attributes of other data types into numbers (e.g. for text – using vector space model). However, this transformation may lead to a loss of some properties of the original data representation (this issue for the vector space model is discussed in Chapter 3-1-2).

Results of different clustering algorithms may have very different properties. Most important are:

- Hierarchical structure of groups

  In most of the cases it turns out that clusters created by an algorithm could be further split into smaller (or merged into larger) ones, depending on how much details do we want. Performing clustering on an example collection of documents concerning intelligence, we could distinguish groups *"artificial intelligence"* and *"emotional intelligence"*. But cluster concerning documents about artificial intelligence could be split further into ones such as *"neural networks"* or *"knowledge discovery"*. Without a hierarchy the clusters would be either too general and containing a lot of documents, or there would be too many of them. Using a tree instead of list of groups enables the user to quickly navigate through it from general to detailed topics.

- Overlapping groups

  Some of the clustered objects can be assigned equally well to *more* than one cluster. Consider for example a great book about UNIX network programming, written by W. Richard Stevens. It belongs to two categories: books about network programming and about the UNIX operating system, and assigning it to only one of them would mean loss of valuable information.

- Leaving of some of clustered objects besides created groups.

  Usually among the data there exists some number of objects that aren't similar to any others. Such elements (called **outliers**) should not be classified as they wouldn't fit to any group.

4.1.1. K-MEANS

K-means is one of classical methods in the cluster analysis ([Rocchio 66], [MacQueen 67]). It is based on numerical data, so in order to employ it to cluster text documents one should use the vector space model. This algorithm is non-hierarchical and in basic version it doesn't allow the clusters to overlap. In order to use k-means, we must first specify the desired number of clusters *k*. Then the algorithm acts in the following manner:

```
Select k initial random centroids
Do
    Assign each object to the closest centroid
    Recompute all centroids
While (!centroids are stabilized)
```

Fig. 4-1-1-1 Outline of the k-means algorithm

This method iteratively optimizes assignment of objects to clusters, using as a criterion distance of each object from the cluster to its **centroid**, computed as mean vectors of objects that belong to respective clusters (therefore centroids are called also **means** and hence algorithm's name).

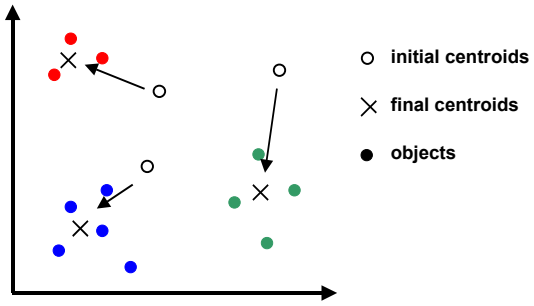

Fig. 4-1-1-2 Example of k-means clustering.

Main advantage of the algorithm is its speed. It requires only *O(nk)* time per one iteration, where *k* is the desired number of clusters, and *n* – number of input documents (when considering documents clustering algorithms, *n* usually denotes size of document collection, so from now on whenever no meaning for *n* is given, it it supposed to have exactly this meaning). Number of iterations is usually significantly smaller than number of objects (although there is no such guarantee), so usually it is said that *overall* complexity of the k-means algorithm is *O(nk)*.

Unfortunately, this algorithm is too simple and so it performs poorly, especially when applied to textual data [Zamir and Etzioni 98]. Its main drawbacks are:

- Necessity of knowing the number of clusters *in advance* (usually there is no way of knowing how many clusters exist).

- The way of initializing centroids is not specified, so they usually have to be randomly chosen. An attempt to overcome both this and former problem by trying to find the initial centroids using other algorithm is discussed in Chapter 4-1-2.
- Only *local* optimization of cluster assignment (a "greedy" method)
- Inability of the method to deal with clusters of non-convex shapes – due to usage of the centroids as cluster representation

There exits also a "fuzzy" version of the k-means algorithm [Bezdek 81]. In this method, each document instead of similarity to the closest cluster has a degree of membership in each cluster so that *overlapping* clusters are allowed:

```
Select k initial random centroids
Do
    Compute for each object its degree of membership in all centroids
    Recompute all centroids
While (!centroids are stabilized)
```

Fig. 4-1-1-3 Outline of the fuzzy k-means algorithm

4.1.2. OTHER METHODS BASED ON THE VECTOR SPACE MODEL

An approach to deal with mentioned earlier problems that arise when using k-means algorithm was proposed in [Cutting et al. 92]. This method – called **Buckshot** – calculates initial cluster centroids applying the AHC algorithm to a (random) small sample of documents of the collection. Then groups found by AHC are taken to be these centroids. This trick allows not only to find the number and position of initial clusters, but also to speed-up computations, as k-means has to perform a smaller number of iterations before stabilization when it knows already the initial centroids. As AHC algorithm used in Buckshot runs in time $O(n^2)$, the initial sample is of size $\sqrt{kn}$ in order to maintain the overall performance of $O(kn)$. A comparison of clustering methods performed in [Zamir and Etzioni 98] shows that using this method improves both time and quality of clustering, but this gain unfortunately is not significant. Main drawback of this method results from the fact that the initial sample may not be representative for the whole document collection.

Another simple approach is the **Single-pass** method [Hill 68]:

```
Treat the first document as the first cluster.
Do
    Add the next document from the list to the closest cluster if their
    similarity is above the similarity threshold (predetermined)
    Otherwise treat it as a new cluster
While (!all documents are clustered)
```

Fig. 4-1-2-1 Outline of the Single-pass algorithm

This method has computational costs similar to k-means, namely it runs in linear time *O(kn)*, where *k* is the desired number of clusters. However, it also has similar disadvantages. Its another weak point is that clustering results depend heavily on the order of documents in the input list. Again according to [Zamir and Etzioni 98] it is the method that has results of the worst quality.

4.1.3. SUFFIX TREE CLUSTERING

Suffix Tree Clustering (STC) is a brand new clustering algorithm, designated only for *documents* clustering purposes. It was proposed originally by Zamir and Etzioni ([Zamir and Etzioni 98]), and is the first clustering algorithm that fulfills their requirements for Web documents clustering methods (presented in Chapter 2.3). It is based on the idea of creating of clusters from documents that share common phrases.

First phase of the algorithm is finding common phrases and documents that share them. It is done using a special data structure called Suffix Tree [Ukkonen 95]. This structure enables us to find all phrases and documents that share them in time linear in the number of documents, and moreover new documents can be added to it *incrementally*, just as they arrive from the search engine. An example suffix tree constructed from three documents is shown below on Figure 4-1-3-1:

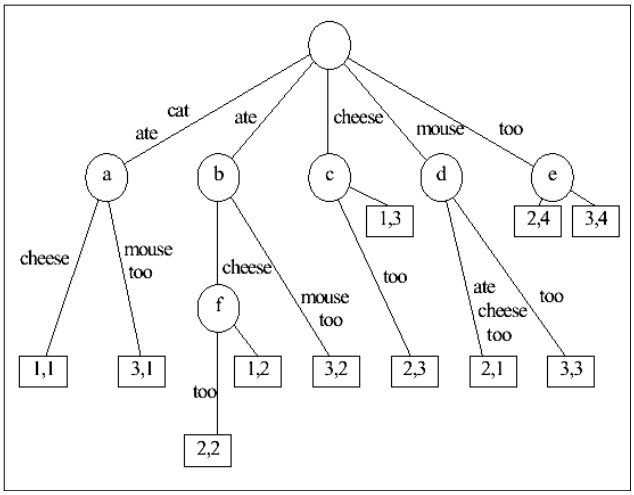

Fig. 4-1-3-1 Sample Suffix Tree for three documents containg sentences: "cat ate cheese", "mouse ate cheese too" and "cat ate mouse too" [Zamir and Etzioni 98].

In the next stage, a list of base (initial) clusters is created. All phrases that occur in more than one document are considered as base clusters. Then for each base cluster, its **score function** is calculated. In essence, this function promotes groups that contain bigger number of documents and are described by longer (i.e. more informative) phrases. Clusters, whose score doesn't exceed certain threshold value are rejected.

Purpose of the last stage of STC is to merge base clusters that share a considerable amount of documents. In this phase, a base cluster graph is created. Nodes of this graph correspond to base clusters, and edges exist between clusters that share more than certain part of documents.

Coherent subgraphs of this graph are said to be result clusters, and phrases associated with respective base clusters are taken to be their descriptions.

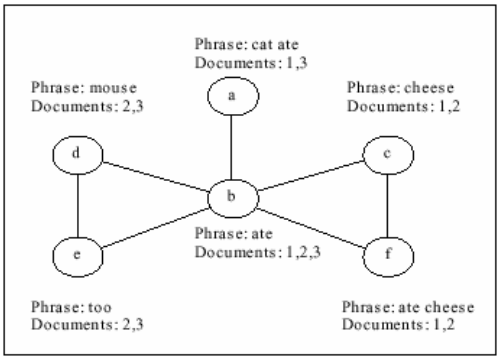

Fig. 4-1-3-2 Base cluster graph for suffix tree from Figure 4-1-3-1 [Zamir and Etzioni 98].

This method was the first really successful search results clustering algorithm. Its authors claim that it is both faster (due to linear complexity and incremental processing) and gives better results than any method based on the vector space model [Zamir and Etzioni 98].

Main disadvantage of basic version of STC is "flat" structure of result clusters, i.e. lack of their hierarchy. An attempt to overcome this problem by creating hierarchy from the result groups is presented in [Masłowska and Słowiński 03].

4.1.4. LINGO

During work on the Carrot^2 system a new clustering algorithm, called LINGO was proposed [Osiński 03]. This method works in a very specific way, namely it firstly finds in the input documents set phrases that would fit to be cluster descriptions (called "abstract concepts") and only then it creates the groups themselves by trying to match documents to the found descriptions. Therefore this approach is called by its author "description-oriented". LINGO is also based on the vector space model, and it employs advanced techniques such as Singular Value Decomposition of the term-document matrix or its dimensionality reduction (see Chapter 3-2-2) using low-rank approximation.

Main advantage of this algorithm are very good, informative to the user labels describing created clusters (however in some cases they are too specific and fit only some of the documents from the group). LINGO is also capable of creating overlapping results. Its main weak points are inability to perform hierarchical clustering and quite slow processing time, due to use of Singular Value Decomposition. However, attempts to accurately assess this method are premature, as it is still a completely new idea, with only one existing implementation.

4.2. THE AGGLOMERATIVE HIERARCHICAL CLUSTERING (AHC) ALGORITHM

4.2.1. ALGORITHM OUTLINE

Agglomerative Hierarchical Clustering is a classical clustering algorithm, originating analogically like k-means from the statistics domain. Fundamental idea of this method is very simple, not to say obvious. It consists in iterative merging of two most similar groups, which in first step contain single elements:

```
Place each object in a separate group
While (!stop condition && groups number > 1)
   -  find 2 most similar groups
   -  link them
```

Fig. 4-2-1-1 Outline of the AHC algorithm

Algorithm takes as input a matrix of pairwise similarities between objects. In case of documents this matrix is created by calculating all pairwise similarities between documents using one of measures presented in Chapter 3-3 (according to our literature review it is usually cosine similarity). It returns a binary (because it *always* links two clusters) tree of clusters, called **dendrogram** (sample dendrograms are depicted on Figure 4-2-2-6). Name of the algorithm refers to its way of working, as it creates hierarchical results in an "agglomerative" or "bottom-up" way, i.e. by merging smaller groups into larger ones. There also exists a "divisive" (rather rarely used) version, which works just oppositely, iteratively partitioning clusters instead of merging them:

```
Place all objects in a single group
While (!stop condition && groups number > documents number)
   -  find 2 most dissimilar groups
   -  split them
```

Fig. 4-2-1-2 Outline of the divisive ("top-down") hierarchical clustering algorithm

AHC is in reality just a *template* of some class of algorithms, with lots of detailed versions proposed in literature. Basic outline of this method is very general and does not specify two main points:

- stopping condition

  Normally, this method stops when only one group is left, so there is nothing left to cluster. But, as this method turns out to be quite slow, it is sometimes suggested that it should be stopped when either specified number of clusters remains or similarity value between two closest clusters in iteration falls below certain threshold. This approach often fails, as these two thresholds have to be arbitrary, and it is hard to find a value that

would suit different cases of input data (simple experiment with manipulation of value of this parameter is described in Chapter 6-3-4). Moreover, usually subsequent iterations of the algorithm take less and less time, so in order to have a significant time gain, the quality trade-off would have to be too big.

- similarity measure between clusters

  This is a very important issue in the AHC method, and may have a great impact on its results. As we have already stated, the algorithm merges in each step two closest clusters. But it receives as input only similarities between *objects*, not *groups*. So there must be some way of calculating the latter. Similarity measures between clusters are called **linkage methods**, as they influence the way the clusters are merged. Different linkage methods are discussed in details in Chapter 4-2-2.

AHC is often considered as algorithm that gives the best results among ones based on the vector space model [Zamir and Etzioni 98]. Its biggest advantage is that it is capable of creating a hierarchy of clusters, which is necessary to reflect the hierarchy of topics in documents. Unfortunately, results of typical version of AHC don't have other properties described in Chapter 4.1 (namely, they cannot contain overlapping clusters, and – when no extra stop condition is used – this method clusters all objects). However, the latter problem can be solved by post-processing of dendrogram given by the algorithm (see Chapter 4-3-5).

But real main disadvantage of AHC is its low speed – it takes between $O(n^2)$ and $O(n^3)$ time to cluster n documents, depending on the linkage method used ([Voorhees 86], [Maarek et al. 00]). This means that using AHC for large document collections may be impossible. Although in Web search results applications the size of input data set amounts usually only to few hundreds documents (see key requirements for Web document clustering methods in Chapter 2-3), we also have less time as clustering has to be performed "on-line".

4.2.2. LINKAGE METHODS

Linkage methods are the most important issue in the AHC algorithm, and in fact they decide how it exactly works. As it was already mentioned in the former subchapter, they are used to aggregate similarities between *single documents* in two different groups into similarities between the both *whole groups*. So linkage methods formulas are based on these inter-document similarities. There are many methods [Everitt 80], and we will describe here only the three most popular of them, presented in [Salton 89]:

- Single-link or Nearest Neighbour

  In this method, similarity between a pair of clusters is taken to be the similarity between the *most similar* pair of objects from each cluster, thus each cluster member will be more similar to *at least one* member of the same cluster than to *any* member of any another cluster:

$$sim(c_i, c_j) = \max_{x \in c_i, y \in c_j} sim(x, y)$$

Fig. 4-2-2-1 Formula for the Nearest Neighbour linkage method

This method merges clusters that have at least one pair of similar items. Unfortunately, it tends to produce a small number of "straggly", large and loosely connected clusters. This is so called **chaining effect** (illustrated also in Figure 4-2-2-6), and happens due to the fact that only two of the elements in cluster have to be very similar. Examples of Single-link clustering results are shown on the below figure:

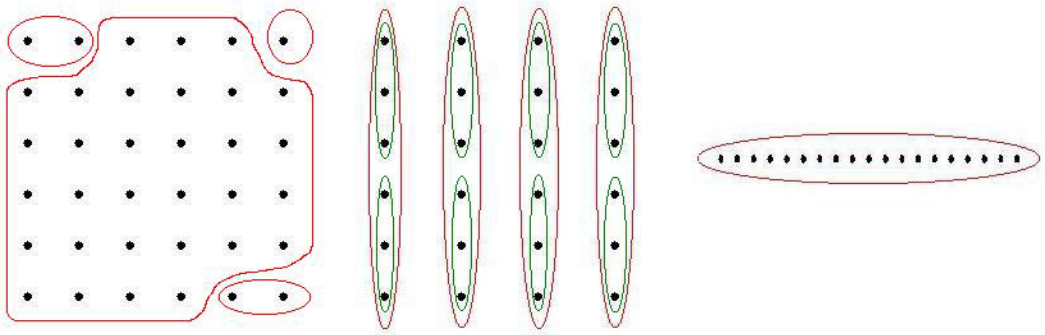

Fig. 4-2-2-2 Three sample results of the Nearest Neighbour linkage method

- Complete-link or Furthest Neighbour

In this method, similarity between a pair of clusters is taken to be the similarity between the *least similar* pair of objects from each cluster, thus each cluster member will be more similar to *all* members of the same cluster than to the *least* similar member of any another cluster:

$$sim(c_i, c_j) = \min_{x \in c_i, y \in c_j} sim(x, y)$$

Fig. 4-2-2-3 Formula for the Furthest Neighbour linkage method

This method merges clusters that have all pairs of their members similar. It produces rather smaller, but tightly connected clusters that are typically preferable. Examples of Complete-link clustering results are shown on the below figure, and it is clearly visible that clusters given by this method make more sense that results of Single-link:

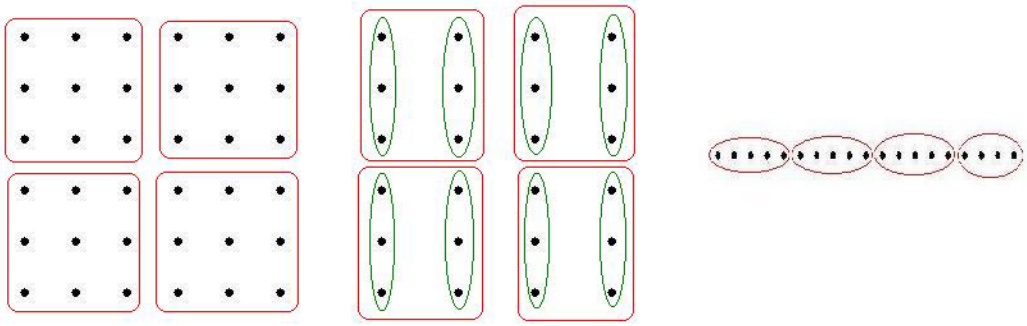

Fig. 4-2-2-4 Three sample results of the Furthest Neighbour linkage method

- Group-average or UPGMA (Unweighted Pair-Group Method using Arithmetic averages)

  In this method, similarity between a pair of clusters is taken to be the *average similarity between all pairs* of objects from each cluster, thus each cluster member will have a greater average similarity to *all* members of the same cluster than to *all* members of any another cluster:

$$sim(c_i, c_j) = \frac{1}{|c_i| \cdot |c_j|} \sum_{x \in c_i} \sum_{y \in c_j} sim(x, y)$$

Fig. 4-2-2-5 Formula for the Group-average linkage method

Example dendrograms for the same documents collection containing 100 items using cosine coefficient as similarity measure between documents are shown below. On the one corresponding to Single-link method, the chaining effect is visible, while dendrogram for Complete-link shows that similarity of larger groups decreases rapidly, and the top, most general groups have their similarity equal to 0. Dendrogram corresponding to Group-average seems to be somehow "intermediate" between two former methods.

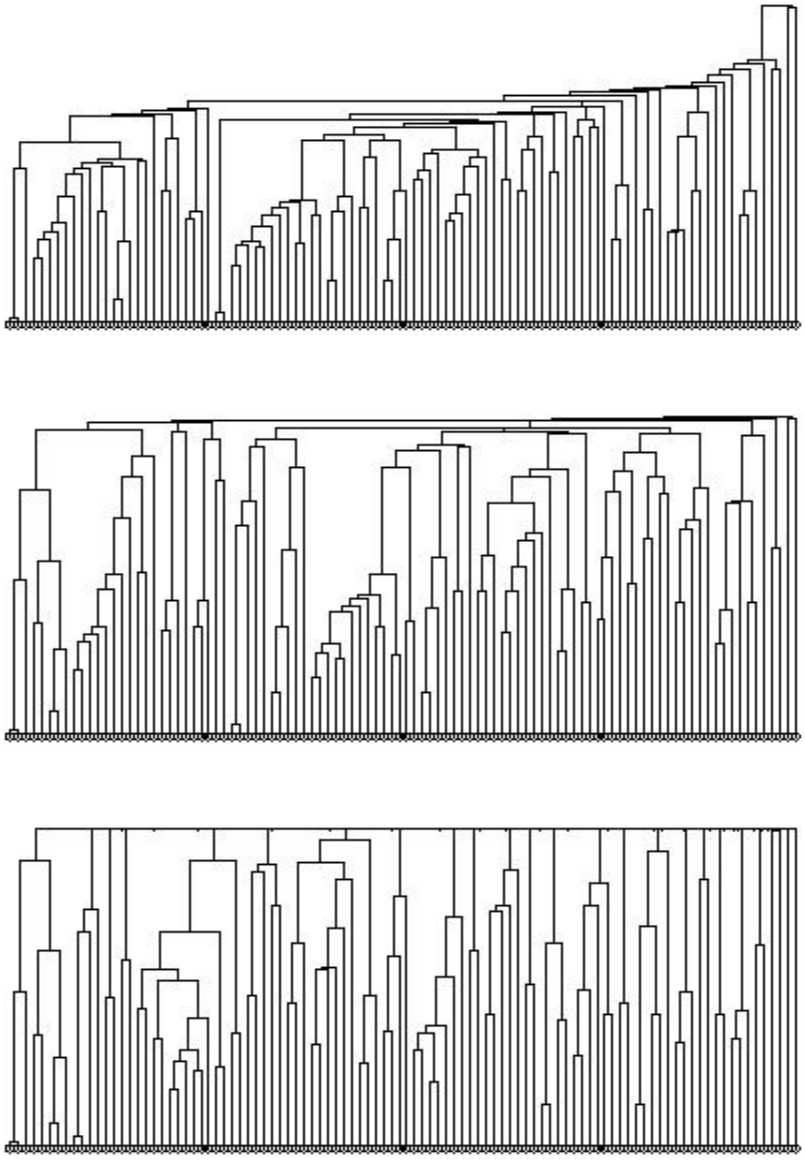

Fig. 4-2-2-6 Example dendrograms created using different linkage methods (from top to bottom: Single-link, Group-Average and Complete-link). Depth of tree node represents similarity between clusters that are leaves of this node.

Formulas used to calculate similarity between groups used in most linkage methods may be reduced to a generic form:

$$sim(a,bc) = \alpha_b \cdot sim(a,b) + \alpha_c \cdot sim(a,c) + \beta \cdot sim(b,c) + \gamma \cdot |sim(a,b) - sim(a,c)|$$

where:

b, c – merged clusters,

a – any other cluster,

α_b, α_c, β, γ – coefficients specific for different linkage methods

Fig. 4-2-2-7 Lance & Williams flexible method

This formula is so called "Lance & Williams flexible method" [Lance and Williams 66]. Sample values of formula coefficients for discussed before linkage methods are presented below.

Single-link:	$\alpha_b = ½, \alpha_c = ½, \beta = 0, \gamma = -½$
Complete-link:	$\alpha_b = ½, \alpha_c = ½, \beta = 0, \gamma = ½$
Group-average:	$\alpha_b = \|b\|/(\|b\|+\|c\|), \alpha_c = \|c\|/(\|b\|+\|c\|), \beta = 0, \gamma = 0$

Fig. 4-2-2-8 Sample coefficients for Lance & Williams flexible method

4.3. FORMING CLUSTERS FROM RESULTS OF AHC

4.3.1. INITIAL CLUSTERS FORMATION

Dendrogram coming as a result of the AHC algorithm consists always of $n - 1$ nodes (for n leaves representing documents). For each of these nodes three cases are possible:

- A node has two leaves. In this case it is a group containing two documents.
- It has a leaf and another node. Then document represented by leaf and group represented by node should then be merged together into one group.
- It has two nodes. It means that it is a group, made of another two subgroups

This idea is illustrated on Figure 4-3-1-1 below:

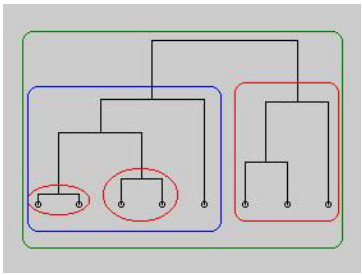

Fig. 4-3-1-1 Example of forming groups from dendrogram

Process of forming clusters from the output tree may be stopped after reaching certain similarity threshold in order to not create large, but loosely-connected groups.

Unfortunately, dendrogram created by AHC apart from informative, coherent clusters contains also a lot of useless, redundant ones, which should be removed from it before presentation of results to the user. Therefore we need to create methods which would enable to sift out only the important clusters from the tree.

The most important source of above problem is the *binary* structure of dendrogram, caused by the way the algorithm links groups – always by two. Unfortunately in the documents collection that might not always be the case (i.e. a topic can have either just one ore more than two subtopics). Consider an example set of documents corresponding to query *"king"*. It

would consist of clusters about Martin Luther King, King's College, King County, King Lear etc. But AHC always links clusters by two, and so it has to create "artificial" groups:

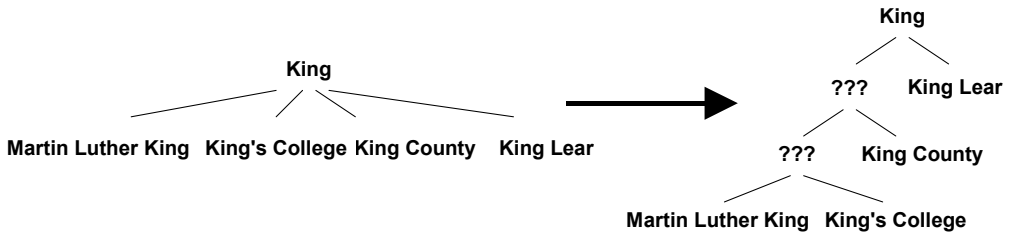

Fig. 4-3-1-2 Example document collection structure (on the left) and corresponding AHC dendrogram. "Artificial" clusters are marked with "???".

It is clearly visible that these useless clusters should be removed, as their presence in the results obscures the hierarchy and makes its browsing by the user much more difficult.

4.3.2. DENDROGRAM POST-PROCESSING

Several methods of dendrogram pruning, mostly based on similarities between the nodes in it have been proposed in order to select the most significant levels of hierarchy. An intuitive approach, whose authors claim that it follows expert's way of most important clusters selection has been proposed in [Maarek et al. 91]. It analyses the changes in similarity values when navigating between different levels of dendrogram. If a group of nodes shares a common parent, and the differences in similarity level between its members are relatively small, then it forms a cluster. This approach is illustrated in Figure 4-3-2-1:

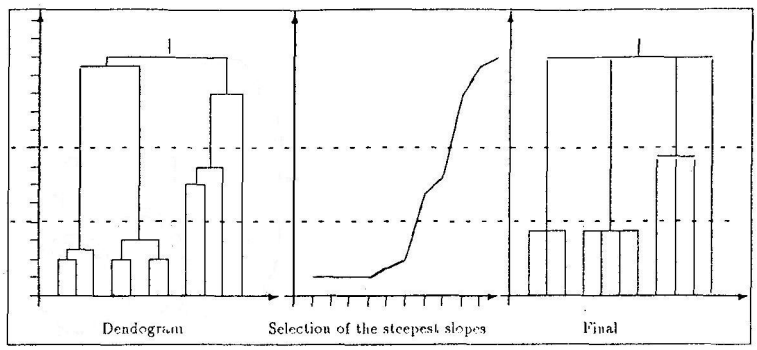

Fig. 4-3-2-1 Example of dendrogram prunning taken from [Maarek et al. 91].

This method can be further simplified as described in [Maarek et al. 00] by rounding the similarities of the dendrogram nodes to some value (authors of this paper find 0.1 to be enough) in order to make the hierarchy more compact. Then all nodes having the same similarity value and a common parent are treated as a cluster. When using Single-link or Complete-link linkage methods, we can round the similarity values in the input matrix, not in the output dendrogram and therefore even remove the post-processing phase from the

algorithm. This is due to the fact that using these linkage methods does not introduce any new values to the similarity matrix during its recomputations by AHC (see Chapter 4-2-2).

However, instead of these approaches we propose a different one, based not on some *numerical* properties of the tree, but rather on *textual descriptions* of created clusters. It is discussed in Chapter 4-3-5, as we would like first to present the way these descriptions are created.

4.3.3. CREATING DESCRIPTIONS OF CLUSTERS

Unfortunately, the AHC algorithm itself contrary to methods like STC or LINGO doesn't create any labels of the formed groups. But when we already have the clusters and their contents, the task of describing them isn't difficult. An intuitive approach is to form description of a group from terms common in documents that are members of this group (we should keep in mind the fact that sharing common terms is the reason why these documents form a group). Detailed version of this method has been presented in [Maarek et al. 00] and is used in our system. It consists of three main points:

- Label of a group consists of several (e.g. 3-5) terms or phrases. The purpose of this rule is simply to constrain the length of descriptions in order to make them more readable for the user.
- Label is made up from the "best" terms or phrases (i.e. ones that have the biggest sum of weights in documents contained in the cluster). When using standard terms weighting methods, it means that words included in descriptions appear frequently in documents. A possible improvement at this stage would be taking into consideration length of phrases (if it is not done already in the terms weighting process – see Chapter 3-5-3). Preferring longer phrases could help to overcome one of weaknesses of AHC, namely short cluster descriptions (refer to Chapter 6-4 for comparison of results given by AHC and other clustering algorithms).
- Terms / phrases from the label have to occur in a considerable part (e.g. 70 %), but not necessarily *all* of documents from this group. It is obvious that terms appearing rarely in the group shouldn't form its description. On the other hand, it would be a mistake to exclude a term from a label if it is not present in just few members of the cluster.

Creating cluster descriptions is this part of the clustering process, where application of phrases gives very good results. According to [Zamir and Etzioni 99], phrases are *"extremely useful in characterizing the cluster's content"*. This issue has been also discussed also in Chapter 3-5-1.

4.3.4. REMOVING REDUNDANT DESCRIPTIONS

Descriptions obtained using method presented in the former subchapter still have some flaws. Usually the problem is that they contain a lot of similar and therefore redundant phrases. Two main reasons of this situation are:

- hierarchical structure of clusters

  Usually set of common terms of a subcluster is made up of a corresponding set of its parent cluster and words specific for this subcluster. So for an example cluster *"King County"*, all its subclusters would have a phrase *"King County"* in their descriptions (i.e. a cluster about government of King County would have a description *"King County"*, *"King County Government"*).

- use of phrases

  As we have already stated in Chapter 3-5-2, during phrases extraction overlapping sequences may be retrieved. For example, in a collection of documents containing phrase *"King County Government"*, algorithm described in this chapter would find also its subphrases: *"King County"*, *"County Government"*, *"King"* etc., and possibly all or most of them would appear in the cluster label, although most of them would be redundant.

Subphrase of a given phrase is defined in [Zamir and Etzioni 99] as a phrase (or term) that is its substring (i.e. two subphrases of *"King County"* would be *"King"* and *"County"*). We propose a more general definition, and call a subphrase of given phrase a phrase (or term) that contains any of terms contained in the first one (but not all of them). This definition is order-independent, so for instance *"agglomerative clustering"* would be a subphrase of *"agglomerative hierarchical clustering"*, while according to the first one, only *"agglomerative hierarchical"* and *"hierarchical clustering"* could be its subphrases. Then **superphrase** of phrase *a* is any phrase *b* such that *a* is a subphrase of *b*. Of course any definition of sub- or super phrases should take into account stemming of terms contained in phrases so that *"King"* would be a subphrase of *"King's College"*.

First problem is simply eliminated by removing terms or phrases that appear in descriptions of parent clusters from descriptions of their subclusters [Maarek et al. 00]. We have enhanced this method by removing also subphrases of phrases that appear in parent clusters labels.

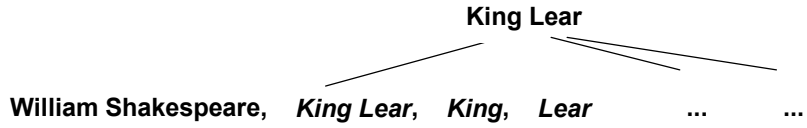

Fig. 4-3-4-1 Example descriptions for subcluster and its parent cluster. Redundant descriptions are written in italics.

The need for this improvement is illustrated on above Figure 4-3-4-1. There we have a cluster of documents corresponding to Shakespeare's *"King Lear"*. It has a subcluster which could be described with following phrases: *"William Shakespeare"*, *"King Lear"*, *"King"* and *"Lear"* as all of these occur frequently in it. It is easy to see that three last phrases are redundant. Using the method from [Maarek et al. 00] we would remove only the phrase *"King Lear"*, but also its subphrases *"King"* and *"Lear"* should be removed from the description.

It should be mentioned here that this method may lead to removal of *all* phrases that make up cluster label. It happens most often in case of "artificial" groups from Figure 4-3-1-2, as they are usually labeled only with phrases that also occur in descriptions of their ancestors in the hierarchy. This is due to the fact that theses groups themselves have no specific meaning.

Second problem is quite common for phrase-based algorithms. Authors of Suffix Tree Clustering have also run into it and have proposed three simple heuristic methods in order to deal with it [Zamir and Etzioni 99]. We have employed here all of these approaches.

4.3.5. DESCRIPTIONS-BASED CLUSTERS POST-PROCESSING

During the work on this thesis we have discovered that in order to perform dendrogram pruning instead of methods already described in literature (see Chapter 4-3-2) one may use techniques based on simple analysis of cluster descriptions. We hold that such approach may give better results, as cluster labels tell us more about meaning of their content than values of their mutual similarities. This approach brings also AHC closer to new generation of clustering algorithms like STC or LINGO, which are based mostly just on group *descriptions*. We will describe here several simple strategies, which are implemented in our system.

- removing clusters without description

  If no description can be created for a group (see Chapter 4-3-3 for information about describing groups), it is removed from the results hierarchy and replaced by its respective subgroups. A rationale for this method is importance of good cluster descriptions (compare with Chapter 2-3). Group with no label would surely be completely *non-informative* to the user, who would have to browse all of its documents in order to deduce its topic.
  This method was introduced mainly in order to deal with "artificial" groups (see Figure 4-3-1-2), for whom usually no description can be created as they correspond to no distinct topics (compare also the former subchapter). It is also the most important of strategies presented here, as it helps to fix our biggest problem.

- merging clusters with the same descriptions

  Sometimes the AHC algorithm creates two very similar, but yet *separate* groups under the same node in the hierarchy. However, the difference between them is not enough to receive different labels in the description process. Then they are merged into one, as description is for us a more important hint than exact dendrogram's structure. Besides, presence of two different groups with the same description might confuse the user.

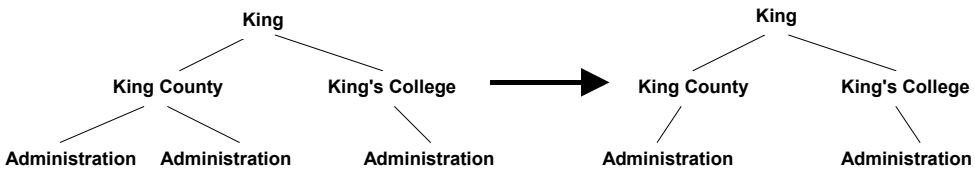

Fig. 4-3-5-1 Example of merging clusters with equal descriptions. Two subclusters of a cluster "King County" are merged as they have the same label. However, they are not merged with subcluster of cluster "King's College" that has the same description, as they have different parent clusters.

- attaching a cluster to another one, whose description is a subset of its description

  This strategy is quite similar to the former one. If two subclusters share a common parent cluster and description of one of them contains all the phrases from description of the second one plus some other phrases, it means that in fact it should be a subcluster of the second one. Moreover, its description is truncated by removing from it phrases common for both clusters. This case is illustrated on Figure 4-3-5-2 below.

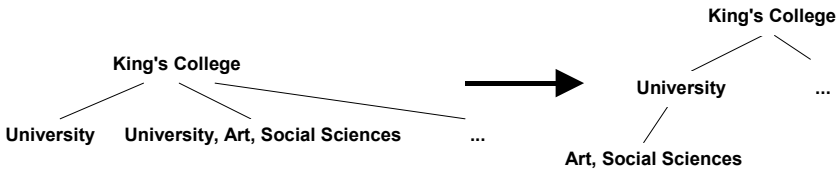

Fig. 4-3-5-2 Example of attaching a cluster to another one with more general description.

As a result of two latter post-processing methods it may turn out that after merging some of the clusters will have only one descendant, and such hierarchy may seem strange. However, we allow such situation, as it may also happen in the input set, when in some group only one subtopics may be distinguished. In such case, the algorithm should reflect this fact in its results.

4.3.6. FINAL POST-PROCESSING

Purpose of methods described in the former subchapter was to choose from the result dendrogram only such clusters, which really correspond to some distinguishing thematic categories. Here we will present three simple approaches used by us to increase readability of the hierarchy to the user:

- removing root of the hierarchy

  If the cluster corresponding to the root of the dendrogram is not removed during pruning phase, then we eliminate it here (independently of its description or any other factor), and its subclusters become the top level of hierarchy. This method is used in order to increase the readability of the first, most important level of hierarchy.

- creating the *"Other Topics"* group

  After post-processing of dendrogram it may turn out that due to removal of some clusters, a certain part of documents from input collection is not contained in any result cluster. These documents usually don't correspond to any of the topics in final clusters structure. A special group labeled *"Other Topics"* is created and they are assigned to it.

- ordering result clusters by their size

  For each of the result groups, its list of subgroups is ordered according to their size, so that the biggest and possibly more important ones appear at the beginning of it.

5. IMPLEMENTATION OF THE AHC ALGORITHM IN THE CARROT^2 SYSTEM

In this chapter we will present the way of implementation of introduced earlier in Chapters 3 and 4 issues, particularly the AHC algorithm. We will also discuss in details already mentioned before Carrot^2 environment, which spared us a considerable amount of work.

Contrary to the previous parts of the thesis, this chapter focuses rather on technical than theoretical aspects of the AHC algorithm. It gives a brief overview of the module's architecture along with its way of working. Moreover, in its final subchapter, issues concerning efficiency of chosen solutions are also discussed.

5.1. THE CARROT^2 SYSTEM ARCHITECTURE

5.1.1. BASIC IDEAS

Carrot^2 [Carrot2] is a generic data-processing framework, developed by Dawid Weiss [Weiss 02a]. It is currently used to perform experiments with different search results clustering algorithms, but as claims its author, *"it can be easily configured to do other, interesting things"*. The system has a pipelined architecture, based on **processes**, being chains of **components**, i.e. *independent* software entities (modules) capable of performing some specified tasks.

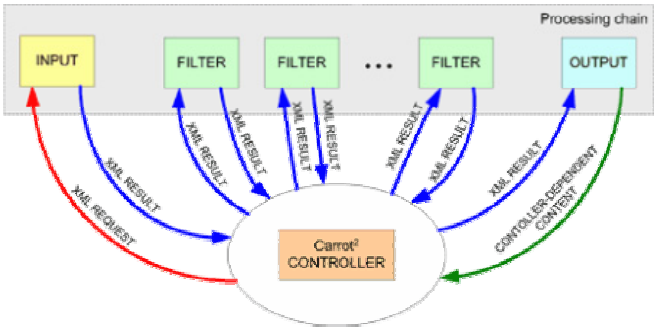

Fig. 5-1-1-1 Components and data flow in Carrot^2 (taken from [Weiss 02a]).

Components communicate between themselves using HTTP protocol instead of function calls, so they may be distributed on different machines. Moreover, the XML language is (usually) used in the inter-component communication. These facts have a number of implications:

- Flexibility, as components are independent software units.

- Easy distribution.
- Low speed of processing due use of HTTP and XML in communication between components. However, use of HTTP enables possible distribution of components which may be an advantage, as it allows load balancing in case of severe network load.

As Carrot^2 was meant to be rather research than commercial, user-oriented system, the main goal was ease of performance of different experiments and not efficiency. Moreover, the clustering process has a pipeline-oriented nature (e.g. *stemming – terms weighting – similarities calculation – clustering algorithm – post-processing* etc.), independently of the type of method used. This architecture completely suits our needs since we can only focus on writing the clustering component and then assemble it in a process with already existing components.

5.1.2. ELEMENTS OF SYSTEM ARCHITECTURE

In the Carrot^2 system, four types of components can be distinguished (see Figure 5-1-1-1):

- input components

  Input component is at the beginning of a process. It accepts a query request from the controller. Then it should generate data corresponding to this query (e.g. download from some search engine snippets relevant to the query)

- filters ("intermediate" components)

  This type of component takes as input some data and transforms it, sending its own results further down the stream to next component in the processing chain. Most important filters in Carrot^2 are so called clusterers, implementing clustering algorithms. Filters associated with the AHC algorithm are presented in Chapter 5-2.

- output components

  Output component converts the received data into a form that may be returned to the outside environment. For instance such component may visualize results of a clustering algorithm as a HTML page.

- controller

  It is the main component in the Carrot^2 system, controlling the cooperation of other ones from the input to the output. It receives the query from the user and returns to him the final data.

Each component is described by a special file, called **descriptor**. It provides all data about a component needed by the controller, namely its network address (needed in order to localize it) and data formats that it produces / accepts. The latter information (not obligatory) enables the controller to check if one component can receive results of another as input.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <service xmlns="http://www.dawidweiss.com/projects/carrot/componentDescriptor"
    framework="Carrot2">
    <!--  XSLT processor   -->
  - <component id="carrot2.output.xslt-renderer.dynamic-tree" type="output"
      serviceURL="http://@DEPLOY_HOST@:@DEPLOY_PORT@/xslt-renderer/service?
      stylesheet=dtree" infoURL="http://@DEPLOY_HOST@:@DEPLOY_PORT@/xslt-renderer">
    - <acceptedFormat>
        <mime>application/x-carrot-clustering-plain-response</mime>
      </acceptedFormat>
    - <acceptedFormat>
        <mime>application/x-carrot-clustering-groups</mime>
      </acceptedFormat>
    - <producedFormat>
        <mime>text/html</mime>
      </producedFormat>
    </component>
  </service>
```

Fig. 5-1-2-1 Component descriptor of a sample output component.

Information about processes is also provided by their respective descriptors. Purpose of a descriptor is to define, which components (and in which order) make up the process and specify values of parameters passed to components included in it. Most important property of descriptors is the possibility to change them at runtime, so one may repeat one clustering process using different values of parameters without either restarting the server or recompiling the components.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <process xmlns="http://www.dawidweiss.com/projects/carrot/processDescriptor" id="carrot2.process.stc-full.google"
    description="Google, English stemming, STC, Dynamic Tree">
  - <processing-script language="beanshell">
    - <![CDATA[

          controller.setUseCachedInput(true);
          controller.setDoCacheInput(true);

          InputStream is;

          is = controller.invokeInputComponent( "carrot2.input.snippet-reader.google", query );
          is = controller.invokeFilterComponent( "carrot2.filter.stemming.en.porter", is );
          is = controller.invokeFilterComponent( "carrot2.filter.stc.full-stc", is );

          Map dtreeOptions = new HashMap();
          dtreeOptions.put("dtree.css", "compact");
          dtreeOptions.put("param.output.terminal.nodes", "true" );
          dtreeOptions.put("param.max.open.level", "0");
          dtreeOptions.put("param.max.subgroup.snippets.included", "3" );
          dtreeOptions.put("param.no.terminal.groups.expansion", "true" );
          dtreeOptions.put("param.snippets.sorting.type", "ORDER_DOCUMENTS_ORDER");
          is = controller.invokeOutputComponent( "carrot2.output.xslt-renderer.dynamic-tree", is, dtreeOptions );

          controller.sendResponse( is );


    ]]>
  </processing-script>
</process>
```

Fig. 5-1-2-2 Process descriptor of a sample clustering process employing STC algorithm.

5.2. AHC MODULE AS A PART OF CARROT^2

Agglomerative Hierarchical Clustering algorithm implementation in the Carrot^2 system consists of two filter components, named TermsFilter and AHCFilter. Main purpose of the first component is to extract terms and phrases from the input snippet list and calculate the term-document matrix (in fact, this matrix is returned by the component as vector of lists due to he mentioned in Chapter 3-1-2 fact that it is sparse). This component performs neither stemming nor stop-words extraction, which have to be done earlier in the processing chain. Example of the output of TermsFilter is shown below:

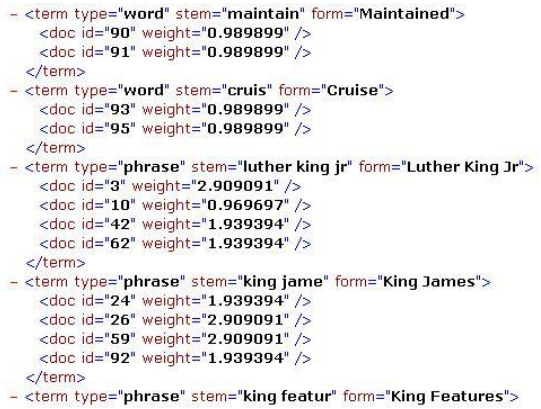

Fig. 5-2-1 A part of sample output of the TermsFilter component.

TermsFilter component receives several parameters that influence its way of working. These parameters listed below:

Parameter	**Value**	**Description**
maxPhrasesLength	integer number (default 5)	maximum length of extracted phrases (refer to Chapter 3-5-2 for details). Setting this value to 1 means that algorithm uses *only terms*
minPhrasesStrength	float number (default 1.0)	minimum strength of extracted phrases (refer to Chapter 3-5-2 for details)
termsWeighing	name of method (default tf weighing)	method used for weighting terms and phrases (refer to Chapter 3-2-1 for details)
strongTermsWeight	float number (default 1.0)	scaling factor for weight of terms and phrases in documents, in whose titles they appear (refer to Chapter 3-2-2 for details)
removeQuery	boolean value (default true)	indicates whether terms and phrases being subphrases of the query should be removed during processing (refer to Chapter 3-2-2 for details)
removeSingleTerms	boolean value (default true)	indicates whether terms and phrases appearing only in one document should be removed during processing (refer to Chapter 3-2-2 for details)

Fig. 5-2-2 Parameters of the TermsFilter component.

AHCFilter receives the output of TermsFilter, calculates matrix of similarities between documents, executes the AHC algorithm and finally performs the cluster labeling along with the dendrogram pruning methods described in Chapter 4-3-5. Its output is the final tree of clusters and their labels, presented below.

Fig. 5-2-3 A part of sample output of the AHCFilter component.

AHCFilter also receives many parameters that enable to change behavior of different parts of the algorithm. However, this variety of parameters may be also viewed as a disadvantage, as it makes tuning of the algorithm very difficult. Parameters of this component are listed below:

Parameter	**Value**	**Description**
similarityMeasure	name of method (default cosine similarity)	method used for calculating inter-document similarities (refer to Chapter 3-3 for details)
linkageMethod	name of method (default complete linkage)	linkage method for the AHC algorithm (refer to Chapter 4-2-2 for details)
stopCondition	name of method (default none)	stop condition for the AHC algorithm (refer to Chapter 4-2-1 for details)
groupsCreatingThreshold	float value (default 0.0)	indicates when initial process of forming of clusters from tree should stop (refer to Chapter 4-3-1 for details)
removeGroupsSimilarWithParents	boolean value (default false)	indicates whether values similarities between dendrogram nodes should be rounded (refer to Chapter 4-3-2 for details)
groupsMergingGranularity	float value (default 0.1)	value to which similarities should be rounded (refer to Chapter 4-3-2 for details)
maxDescritptionLength	integer value (default 3)	maximum number of terms and phrases in the cluster description (refer to Chapter 4-3-3 for details)
minDescriptionOccurrence	float value (default 0.6)	minimal occurrence of phrase or term from a cluster description in this cluster (refer to Chapter 4-3-3 for details)
groupOverlapThreshold	float value (default 0.5)	(refer to [Zamir and Etzioni 99] for details)
groupCoverageThreshold	float value (default 0.2)	(refer to [Zamir and Etzioni 99] for details)

removeGroupsWithoutDescription	boolean value (default true)	indicates whether clusters without descriptions should be removed (refer to Chapter 4-3-5 for details)
mergeGroupsWithSimilarDescriptions	boolean value (default true)	imdicates whether clusters with the same descriptions should be merged and clusters having descriptions that are supersets of descriptions of other clusters should be attached to them (refer to Chapter 4-3-5 for details)
removeTopGroup	boolean value (default true)	indicates whether root of the dendrogram should be removed (refer to Chapter 4-3-6 for details)

Fig. 5-2-4 Parameters of the AHCFilter component.

Main rationale for splitting of the AHC module into two ones was fact that the first phase of algorithm (i.e. terms weighing) may be common for many clustering methods. So once implemented, TermsFilter component can be reused many times in different processes.

Unfortunately, splitting of this module has caused a slight increase of its execution time (efficiency issues will be discussed further in more details in Chapter 5-3-2). Output of the TermsFilter component is directly passed to input of the AHCFilter component. So if they were two classes in *one* software unit, passing of data between them could be realized simply just by passing of few references to memory. But as they are two *separate* components, we have to store and then read whole lists of hundreds of objects (all input documents and terms and phrases occurring in them with their respective weights) in XML format. However, influence of these tasks on the overall module's performance is not very significant, as they take just a dozen or so percent of its total execution time (when clustering 100 snippets), and even less for larger amounts of data (compare Figure 5-3-2-2).

5.3. AHC MODULE DESIGN AND IMPLEMENTATION

5.3.1. GENERAL DESIGN AND IMPLEMENTATION ISSUES

Both components making up AHC module were designed in the UML modeling language [UML], being a *de facto* standard in object-oriented modeling. Due to definitely research nature of the project and frequent changes of requirements, only a rough, general design was created, consisting of the following diagrams:

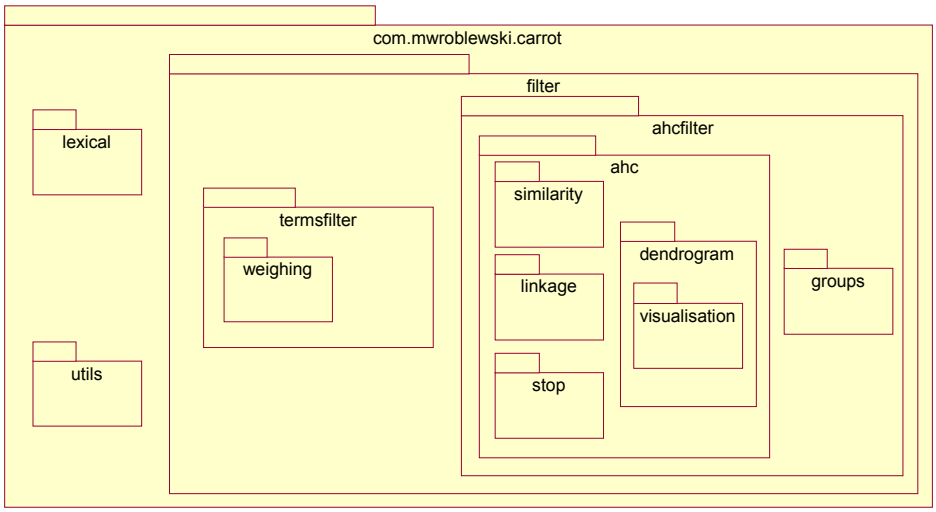

Fig. 5-3-1-1 General architecture of the AHC module in the Carrot² system on UML component diagram. Two main packages: `termsfilter` *and* `ahcfilter` *correspond to filter components of the Carrot² system.*

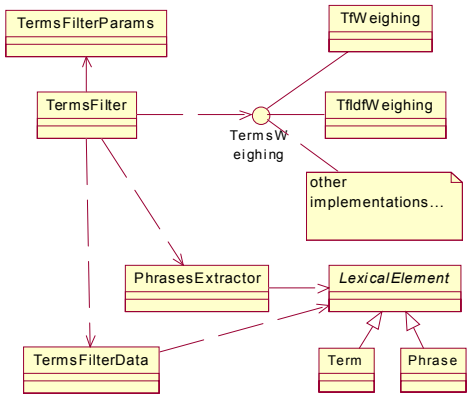

Fig. 5-3-1-2 General class diagram for the `termsfilter` *module*

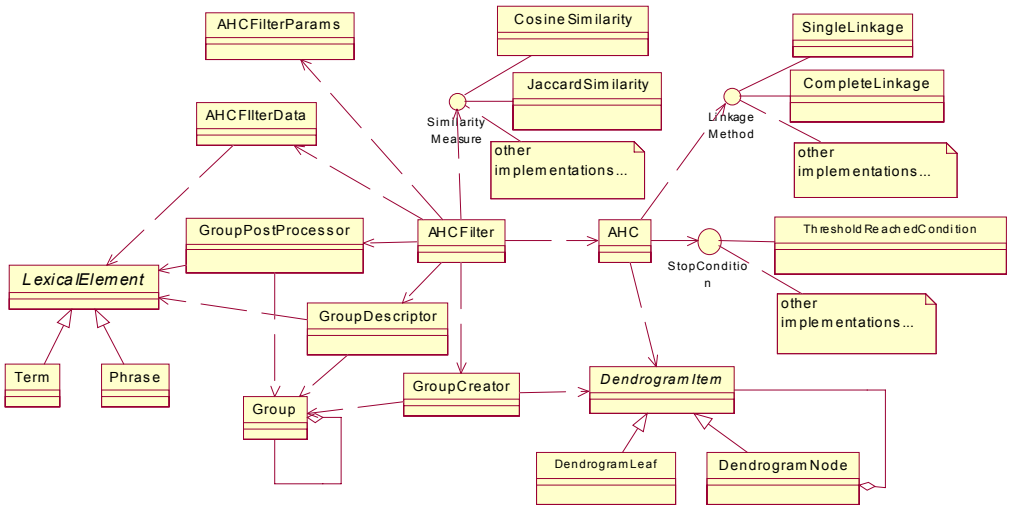

Fig. 5-3-1-3 General class diagram for the `ahcfilter` *module.*

It was already mentioned that one of the major goals during design of the system was ease of performing different experiments. Due to this fact, we have put great emphasis on parametrization of the module. Many tasks, such as for instance terms weighting or inter-document similarities calculation may be performed in may different ways. Therefore for each of them several different classes having a common interface that may perform the same task were implemented (see Figures 5-3-1-2 and 5-3-1-3). As objects of these classes may be passed as parameters to respective components in descriptor, it is possible to replace one method of for instance term weighing by another during runtime.

As the language of implementation, we have chosen Java, being also the language in which is written the whole Carrot^2 system (although its architecture in fact imposes no constraints on used language [Weiss 02a]). Java was chosen because it has a number of very useful features, namely:

- A number of most important data structures such as lists or hash tables are already implemented in the standard libraries of the language.
- Java language itself enforces object-oriented-programming and good software practices.
- Java Virtual Machine performs runtime checks for common and often difficult-to-track errors such as null pointers, array indices running out of bounds, etc.
- Many libraries supporting Web technologies (particularly XML) written in that language.
- A number of utilities for the Carrot^2 system (component frameworks etc.) was already implemented in this language.

Serious drawback of Java is unfortunately speed of execution of code written in this language (much lower than in case of natively-compiled code, written for instance in C++). However, as we have stated before in Chapter 5-1-1, due to character of the system, speed was not the most important criterion in the choice of solutions. Efficiency issues are discussed in details further in the next subchapter.

5.3.2. EFFICIENCY ISSUES

Unfortunately, speed of execution of our AHC module is quite slow. Performed tests showed that for more than just 200 snippets system's time of response exceeds several seconds, making it too slow for more impatient users. Exact results of these tests, with values for individual parts of modules are given below in Figure 5-3-2-1 (please have in mind fact that these are just times of the AHC module, not the whole Carrot^2 system):

Number of input snippets	TermsFilter				AHCFilter					Whole module
	input phase	terms & phrases extraction	terms & phrases weighting	output phase	input phase	similarities calculation	the AHC algorithm	dendrogram prunning	output phase	
100	0.073	0.584	0.010	0.082	0.096	0.055	0.236	0.065	0.027	1.227
200	0.125	0.995	0.022	0.313	0.187	0.451	1.189	0.117	0.038	3.437
300	0.308	2.001	0.032	0.415	0.386	1.357	4.009	0.260	0.067	8.834
400	0.216	3.163	0.046	1.059	0.400	3.918	10.935	0.741	0.072	20.548

Fig. 5-3-2-1 Running times (given in seconds) of differents parts of the AHC module for the query "king" depending on number of documents clustered (Intel Pentium III 800Mhz machine running under RedHat Linux 7.2 was used).

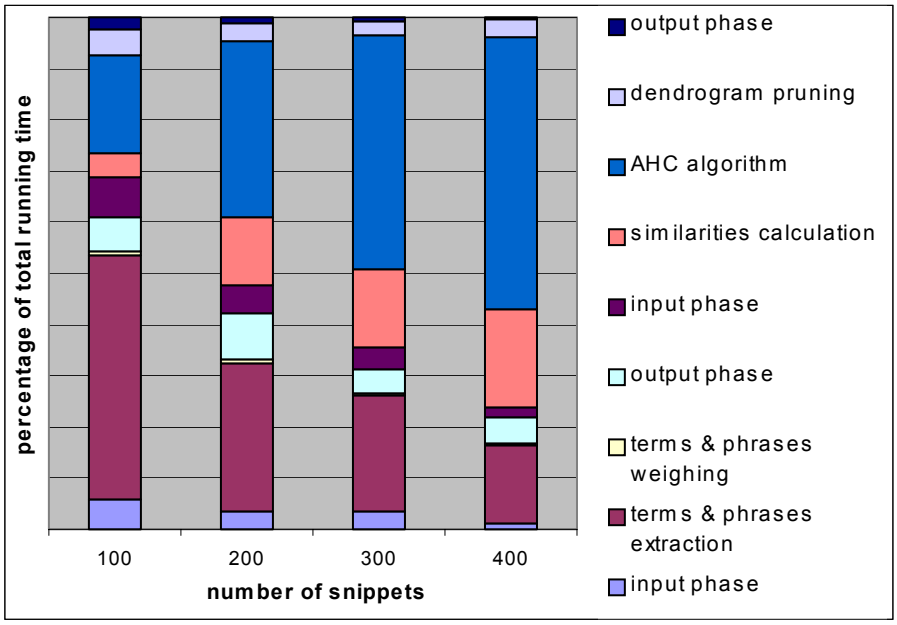

Fig.5-3-2-2 Processing time of different parts of the AHC module as percentage of the whole module's processing time for the example from Figure 5-3-2-1.

It is easy to see that the most laborious element of the module is implementation of the AHC algorithm (only for small input data sets, containing less than 200 snippets extraction of terms and phrases lasts longer than this task), due to its computational complexity. Major goal of implementation of this algorithm in the Carrot^2 system was to make it as simple and as flexible as possible. We have created a general framework, which enables easy replacing of stop conditions and linkage methods. Unfortunately, chosen solution has also the biggest complexity possible for this algorithm – $O(n^3)$ (see Chapter 4-2-1), being definitely too slow to meet Zamir and Etzioni's key requirements for Web documents clustering methods (presented in Chapter 2-3).

Another element that takes up bigger and bigger part of the whole running time with increase of the size of data is calculation of inter-document similarities. Usually it is said that this process takes $O(n^2)$ time for n documents [Salton 89]. This formula doesn't take into

account the fact that in case of all commonly used similarity measures time needed for their calculation also depends linearly on the number of terms in the input document set. So exact computational complexity of calculating document similarities is $O(n^2 \cdot m)$, where m is the number of terms, which increases with the number of documents (a fact not concerned by most of the authors). However, this increase is slower than linear (there is no exact formula – at least we haven't found any), so overall complexity of this process is between $O(n^2)$ and $O(n^3)$, being just a little bit smaller than that of the AHC algorithm.

6. EVALUATION OF THE AHC ALGORITHM

Purpose of this chapter is to present an assessment of our version of the AHC algorithm. In order to achieve this goal, we have carried out several experiments, whose results are also presented. In this part of the thesis we refer often to the former chapters, showing influence of described there techniques on the quality of the algorithm's results. We have also performed a comparison between our implementation of AHC and several other important clustering algorithms.

Our experiments may be distinguished into two classes on the ground of the way of results evaluation:

- user evaluation (presented in Chapter 6-2)
- expert evaluation (presented in Chapters 6-3 and 6-4)

Major goal of the first type of experiment was to determine the quality of results given by the system from the *common user's* point of view. In order to achieve this goal, we have performed a survey on a group of several users, asking them to judge simple aspects of the algorithm's results. We have also created few simple metrics (presented in Chapter 6-1-2 and Chapter 6-1-3) that give exact, numerical values based on these users assessments.

On the other hand, experiments of the second kind consisted of comparing the results of different algorithms (or one algorithm for different parameters / input data) by a *single* person (expert) in order to find differences between them. In this part of evaluation we have concentrated our attention rather on finding and describing the properties of the algorithm's results than on attempt to calculate some exact metrics. Therefore, this part of the evaluation experiment was more research- than user-oriented, trying to find *reasons* behind the final results, not to *judge* them definitely.

6.1. METHODS OF CLUSTERING RESULTS' EVALUATION

It turns out that evaluation of clustering results (especially in case of hierarchical algorithms) is quite a difficult task. It is particularly hard to create objective, numerical measures of their quality. Here we point out a few most important reasons of this fact:

- subjectivity of evaluation

  The primary criterion in clustering evaluation should be *usefulness* of the given results. However, for such a definition of quality it is hard to find *objective* measures. Different end users may have different preferences (e.g. some may be interested in small, precisely described clusters while others prefer large, more general groups), therefore there may

exist many equivalent ways of partitioning the input documents set. This is a common problem in the Web Mining, as this domain is very user-oriented.
It turns out that there are at least two benefits of clustering – enabling the user to find faster the relevant information and "search refinement", consisting in creating a good partitioning of the input documents set, enabling to find different subtopics in the query results. The used similarity measures should be different depending on which application is more important.

- multi-criteria nature of the problem

Apart from major goal of our evaluation, the quality of results is influenced by a number of factors such, as conciseness of created clusters or eligibility of their descriptions, and it is hard to aggregate them all to *one* factor. This issue will be discussed also in Chapter 6-1-2.

- difficulty in comparing results given by different algorithms

As we have mentioned before in Chapter 4-1, results of different clustering algorithms may have different properties (i.e. be hierarchical or flat, overlapping or not). It is difficult to create quality measures that could be *equally well* applied to different algorithms and therefore *fairly* compare their results.

6.1.1. EXISTING WEB MINING METHODS

Problem of the results evaluation is quite often addressed in the Web Mining literature. During our studies of publications from this domain we have come upon several different approaches and in this subchapter we make a brief overview of them, pointing out both their strong and weak points.

- standard Information Retrieval metrics

Most commonly used quality measures in the Information Retrieval domain are **precision** and **recall** [Salton 89]. These two metrics were originally defined for *search* results evaluation problem as following:

Let:
r – number of documents relevant to the query in the whole document set
a – number of documents relevant to the query retrieved by the algorithm
b – number of documents non-relevant to the query retrieved by the algorithm

Then:

$$precision = \frac{a}{a+b}; \quad recall = \frac{a}{r}$$

Fig.6-1-1-1 Definition of precision and recall.

Both metrics are different and *opposing* quality criterions, as usually the more precise the search results are, the less (also relevant) documents we get. However, these quality measures in their basic version don't satisfy our needs stated before.

- merge-then-cluster approach

Technique that seems to be a natural approach to our problem is performing clustering on a specially prepared set of documents. This input set is created by collecting documents concerning different topics. Then the results of clustering algorithms performed on this set are compared with its original partition [Zamir 99]. The most important advantage of this approach is ability of *automatic* repeating of multiple tests of different methods (or one method with different parameters) after establishment of the input set.
This approach requires the construction of the input documents set to be done very carefully. In particular, one cluster appearing in results of some algorithm may be split by other into several more specific ones, and decision which partition is correct is often arbitrary. One may also criticize the "artificial" way of creating the test documents set by merging documents concerning different topics instead of using real data (ranked lists), in which case finding coherent clusters may be more difficult.

- ground truth hypothesis

Another, but somehow similar approach to this problem is using a quality measure based on entropy ([Maarek et al. 00], [Stefanowski and Weiss 03]). In this approach, the deviation of the algorithm's results from **ground truth** set, representing some "*ideal*" partition is calculated. Constructing the ground truth set may be performed for instance manually by experts who try to find clusters in the input data set [Weiss and Stefanowski 03]. The deviation is calculated basing on a information theory measure presented in [Dom 2001]. This method is similar to merge-than-cluster as the subject of evaluation in both approaches is difference between some reference set and algorithm's results.
It also enables us to quickly perform many different tests after creating reference clustering results (the ground truth). Its main disadvantage is that used quality measure is defined only for flat partitioning (so neither hierarchical nor overlapping results may be correctly evaluated using this method).

- user evaluation

There are many evaluation methods based on the examination of user's behavior. These methods may either make use of direct feedback from users (e.g. some kind of survey), or attempt to analyze automatically their reaction to the system's response for instance by logs tracing. Very interesting and detailed example of user's evaluation is presented in [Zamir and Etzioni 99], where authors of this paper present application of user behavior analysis to evaluation of the Grouper system.
This approach seems to be very promising as the quality of results is decided by the users themselves. However, there are several problems associated with performing of this type of evaluation such as different level of users experience or finding a statistically

significant number of experiment participants [Weiss 01]. Moreover, such experiments take quite a lot of time and cannot be easily, automatically repeated.

6.1.2. OUR CLUSTERING RESULTS QUALITY MEASURES PROPOSAL

During the work on the Carrot^2 system we have decided to make use of the last approach presented in the former subchapter and base system evaluation on user survey. The same method was also used to evaluate results of the LINGO algorithm (see [Osiński 03]), also a part of the Carrot^2 system.

The survey has been created taking into account key requirements for Web search results clustering, such as browsable summaries of the clusters or relevance (refer to Chapter 2-3 for more details). Users taking part in the evaluation were asked to answer the following questions:

- usefulness of created clusters

  For each of the created groups (except the *"Other Topics"* group), the users were asked to assess its usefulness, defined as user's ability to *deduce* topics of documents contained in it on the basis of its *description*. It is easy to see that the most important factor influencing evaluation of this criterion is the algorithm's ability to create meaningful, accurate cluster labels.
  We have to add here that not only the clusters corresponding to topics, in which the user is interested are useful. Provided their descriptions are concise and accurate, also clusters "unexpected" in the context of the query that contain documents irrelevant to the user's interests are also helpful, as they enable him to quickly *separate* these documents from the interesting ones. This is exactly the way, in which clustering improves the search results.
  In case of hierarchy also usefulness of subclusters is evaluated, *independently* of their parent clusters, as top-level clusters that are too general and therefore meaningless may be split into more detailed, useful ones (we continue this topic in the next subchapter).
  Scale: useful cluster | useless cluster

- precision of assignment of documents to their respective clusters

  For each document assignment to a group, the users were asked to judge if this assignment is correct (i.e. whether its snippet fits this group's description). Here we have decided to use a more diverse than binary scale, as documents may fit to some degree in their clusters (e.g. be more or less specific than respective clusters, but still somehow related to their descriptions).
  Precision of assignment is assessed only if the cluster to which the document has been assigned has been judged as useful. There are two rationales behind this decision. First of all, if group's description is nonsense, it is also nonsense to judge if a document fits it. Secondly, if user cannot understand the cluster's title, he will probably discard it and therefore he won't browse documents contained in it. Moreover, the *"Other Topics"* group with its documents is also excluded from this process.

Here we have to make an important remark as in case of both overlapping and hierarchical results one document may be contained in *many* clusters. Therefore its assignment to *all* the clusters where it is contained should be *independently* judged.

Scale: well matching | moderately matching | not matching

Having collected assessments of single groups and documents from users, we may try to define general quality measures for the whole clustering results. We have decided to use *several* metrics evaluating different aspects of the results due to the mentioned earlier multi-criteria nature of the problem. Some of measures used in AHC evaluation were also used to assess the quality of LINGO and are presented already in [Osiński 03]. Here we will focus rather on presentation of AHC-specific solutions, to which we dedicate whole next subchapter.

After collecting of the forms from users we are able to calculate the following statistics:

g – total number of clusters in the results
g_u – total number of useful clusters
s – total number of snippets in the results
s_a – total number of snippet assignments in the results
s_w – total number of snippets well matching their clusters
s_m – total number of snippets moderately matching their clusters
s_n – total number of snippets not matching their clusters
s_o – total number of snippets in the *"Other Topics"* group

On the basis of these statistics, the following quality measures have been defined:

- cluster label quality

  It is the proportion of number of useful and number of all groups (without the *"Other Topics"* group:

  $$\textit{cluster label quality} = \frac{g_u}{g}$$

- snippets assignment distribution

  This measure consists in fact of three ones, describing proportions of number of snippets well / moderately / not matching their clusters and total number of snippet assignments (excluding ones to groups judged as useless and the *"Other Topics"* group):

  $$\textit{fraction of well matching snippets} = \frac{s_w}{(s_w + s_m + s_n)}$$

  $$\textit{fraction of moderately matching snippets} = \frac{s_m}{(s_w + s_m + s_n)}$$

  $$\textit{fraction of not matching snippets} = \frac{s_n}{(s_w + s_m + s_n)}$$

- assignment coverage

  Assignment coverage is the fraction of snippets assigned to clusters other than the *"Other Topics"* group (including also *useless* ones) in relation to the total number of input snippets:

$$assignment\ coverage = \frac{s - s_o}{s}$$

6.1.3. AHC-SPECIFIC QUALITY MEASURES

Presence of hierarchy in the clustering results makes their evaluation even more difficult. Groups included in the results of AHC may contain not only documents, but also other groups, which may also contain other groups and so on. In this connection a number of problems arise, for instance:

- should hierarchical dependencies between groups influence assessment of their quality, e.g. can a subcluster of some useless cluster be useful?

  It seems that top levels of the hierarchy are the most important ones, as their descriptions give to the user the "first impression" of their contents. So even if a useless top-level cluster contains some interesting subclusters, discouraged user may disregard the whole branch of the hierarchy without looking further down into its structure.

- should the redundancy of groups be punished?

  It happens that some levels of hierarchy introduce no distinct subtopics when compared to topic of their parent clusters. In such case, these subclusters should be assessed as useless.

- should the descriptions of parent groups be taken into consideration when evaluating their subgroups?

  It seems that when assessing subclusters descriptions one must take into consideration also their parent clusters description. Consider for instance again results of clustering for a query *"king"*. It may contain a top-level cluster described as *"King County"* with a subcluster *"Government"*. The latter group contains of course documents about government of King County, not about *any* government (in which case it would be too general). On the other hand, during our evaluation experiment we have come into an example hierarchy for a query *"volleyball"* containing a cluster *"women's volleyball"*, which in turn contained a subcluster *"men's volleyball"*. Although the latter one is useful itself, its position in hierarchy makes it rather useless.

In order to at least partially deal with these problems, to evaluate AHC results we have introduced few improvements and additional metrics:

- cluster label quality including size of assessed clusters

  We propose to calculate the cluster label quality factor not on the basis of *number* but *size* of useless clusters (i.e. number of documents contained in them), as the more documents are assigned to useless groups, the more information is lost for the user.
  This factor may be calculated using number of different snippet assignments in the following manner:

$$cluster\ label\ quality = \frac{s_a - s_o - (s_w + s_m + s_n)}{s_a - s_o} = 1 - \frac{s_w + s_m + s_n}{s_a - s_o}$$

- calculating cluster label quality and snippet assignment distribution *separately* for the top level groups

  Apart from calculating these coefficients for *all* clusters, when evaluating hierarchical clustering results one may calculate them only for the *top level* clusters (i.e. ones that have no parent groups). This should be done due to the mentioned earlier fact that top groups are the ones most important to the user.

- average hierarchy depth

  This metric does not measure directly the quality of results itself, but has rather an informative purpose. However, we intuitively feel that the constructed hierarchy has certain optimal depth and shouldn't be neither too shallow nor too deep. In [Maarek et al. 00] dependence of results quality on hierarchy depth for a sample documents collection is shown, but authors of this paper don't give any general conclusions. Optimal hierarchy depth may heavily depend on structure of the input data.

6.2. USER EVALUATION OF THE AHC ALGORITHM

6.2.1. THE EXPERIMENT

Our user evaluation experiment has been conducted on 6 users, who were asked to fill in survey forms. Such a small number of persons taking part in this process may of course seem insufficient in order to recognize its results as statistically significant. Unfortunately, considering our situation, and especially lack of resources and time, it was hard to find a larger number of people that could help us in carrying out the evaluation.

We should also mention that all persons engaged in this process are experienced Internet users, professionally connected with computing science. Moreover, most of them are also experts in the domain of clustering algorithms. This fact had also to influence obtained results.

For the purpose of evaluation we have used four queries, two in both Polish and English languages in order to examine the difference in behavior of AHC for different languages. As we also wanted to study the influence of breadth of topics scope in the input documents set on results quality, two of the queries used in the experiment were general, while another two

were more specific ones. Profiles of queries used in the experiment are given in the below table.

Query	Language	Meaning (for queries in Polish language)	Type	Number of returned snippets
mp3	English	-	general	100
"unix network programming"	English	-	specific	91
siatkówka	Polish	*volleyball*, also *retina* (although much less common)	general	100
aukcje + "aparaty fotograficzne"	Polish	*auctions + cameras*	specific	91

Fig. 6-2-1-1 Queries used in the AHC results evaluation experiment.

Snippets used as input data for the experiment were downloaded from the Google search service. Results handled to the persons involved in the evaluation were obtained using the Carrot^2 AHC module with its default parameter values (mentioned in Figure 5-2-4 and Figure 5-2-2).

6.2.2. RESULTS

Here we would like to present the most important results of the user evaluation. As it has turned out values of quality measures calculated only for top level clusters differ on the maximum only a few percent from their equivalents calculated for the whole hierarchy (a fact somehow unexpected), we present here only values of the latter ones on one, collective chart.

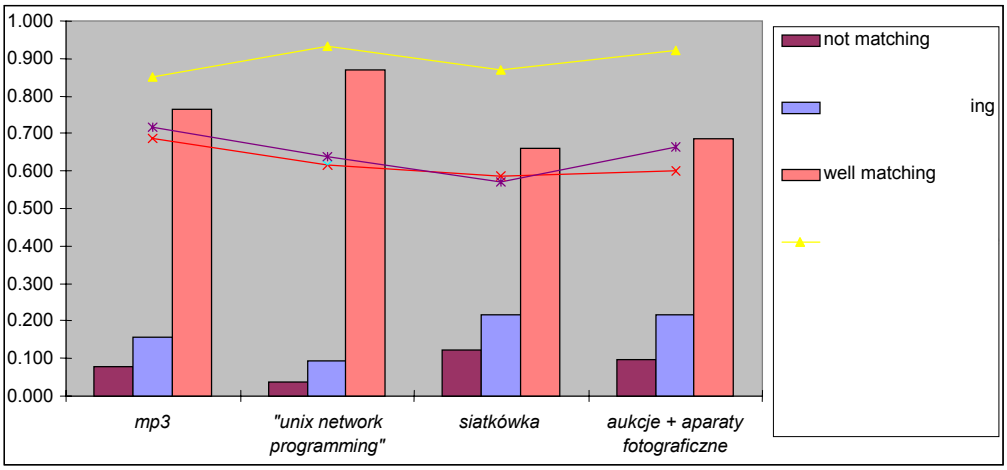

Fig. 6-2-2-1 Evaluation results (snippet distribution, cluster label quality and assignment coverage) across queries.

One may notice that algorithm gives result of quite solid quality. Percentage of well and moderately assigned snippets is high and varies about 90 %, a result that may be considered as satisfying. Unfortunately, AHC deals much worse with cluster descriptions – only about 60-70% of groups have been considered as useful. On the other hand, almost all snippets are

assigned to groups, with value of assignment coverage factor amounting usually to about 90%.

It is easy to see that AHC deals better with results of queries given in the English than Polish language (probably due to less complex issues of stemming and phrases extraction in the case of the first one). Yet influence of query scope on clustering quality is less obvious. It seems that for specific questions snippet assignment coverage is higher and its distribution is better, but when comparing cluster labels quality no general trend can be observed. We must be aware of the fact that our conclusions may lack solid basis, as the experiment was performed on small number of both users and queries.

It is very interesting to compare values of quality measures assessed by different evaluators. As it turns out, user judgements are very subjective, especially in case of cluster label qualities. This simple comparison shows, how difficult is the evaluation of clustering results, in particular in case of hierarchy (see Figure 6-2-2-2 below).

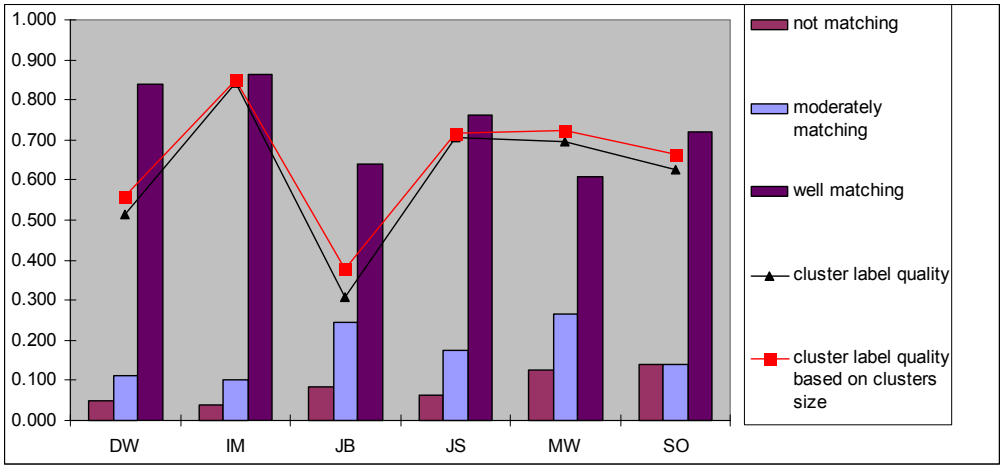

Fig. 6-2-2-2 Evaluation results (snippet distribution, cluster label quality and assignment coverage) across evaluators.

Finally, we would like to compare values of our metrics for results of AHC and LINGO (for detailed description of LINGO evaluation refer to [Osiński 03]). In the below table we have set aggregated results of one algorithm's evaluation against results of another one.

measure	**AHC**	**LINGO**
cluster label quality	0.62	0.76
snippet assignment coverage	0.89	0.72
well-matching snippets percentage	0.74	0.75
moderately-matching snippets percentage	0.17	0.14
non-matching snippets percentage	0.08	0.11

Fig. 6-2-2-3 Comparison of aggregated evaluation results of AHC and LINGO

This juxtaposition shows that while snippet assignment distribution in case of both algorithms is actually the same, LINGO creates significantly better descriptions, at the cost of assigning a lower number of snippets to clusters. It may seem interesting that difference between values of two latter measures for both methods are almost equal (about 15%). So it may seem that this fraction of the input collection is made up of documents that are difficult to place in any cluster, yet the AHC algorithm tries to do just so.

6.3. EXPERT EVALUATION OF THE AHC ALGORITHM - INFLUENCE OF AHC PARAMETERS AND INPUT DATA ON THE RESULTS

Major goal of experiments performed in this subchapter is presentation of the influence of changes in algorithm parameters and input data on its results. Due to already mentioned variety of parameters that may be used in our implementation of the AHC (compare Figure 5-2-4 and Figure 5-2-2), it was impossible to repeat user evaluation similar to one presented in the former subchapter for each studied aspect of the algorithm. We have therefore decided to present the results of these experiments (unfortunately, for large hierarchies we were forced to present only the biggest and topmost groups), with simple verbal description pointing out the most important differences.

During expert evaluation we have used default values of the algorithm parameters from Figure 5-2-4 and Figure 5-2-2 (unless it is mentioned differently in the experiment's description). All queries were formulated in the English language, and we have used top 100 snippets returned by the Google search service as the input data.

6.3.1. INFLUENCE OF QUERY BREADTH

Purpose of this experiment was to compare hierarchy created by the algorithm for two different queries, one general and another more specific, covering a small subset of documents corresponding to the first one. We have chosen two queries: *"clinton"* (it is a widely used query in examples of results clustering as it covers a lot of completely different topics – see also [Zamir 99], [Weiss 01] and [Osiński 03]) and *"president bill clinton"*.

Fig. 6-3-1-1 Comparison of AHC results for queries: "clinton" (general – on the left) and "president bill clinton" (much more specific – on the right).

Result of this comparison shows that the algorithm performs better with more general queries. Main reason for this fact is of course that it is much more difficult to find distinct topics in a set of documents relevant to a specific query, and therefore create good, heterogeneous clusters. In particular, group descriptions are much better when clustering results of general queries.

6.3.2. INFLUENCE OF THE INPUT SNIPPETS NUMBER

Natural influence of increase of input data size should be increase of number of clusters in the results and also increase of their size. It should be also expected that it will improve clusters labels quality, as small documents set may contain proportionally many overspecialized phrases that will be chosen as clusters labels. On the other hand, in case of larger input collection we may expect that these phrases will be replaced by ones a little bit more general, but instead much more informative to the user (while their subclusters may have specific labels). Another important, yet not considered here aspect of increase of the input data size is its influence on the algorithm's processing time. This issue is discussed in more details in Chapter 5-3-2.

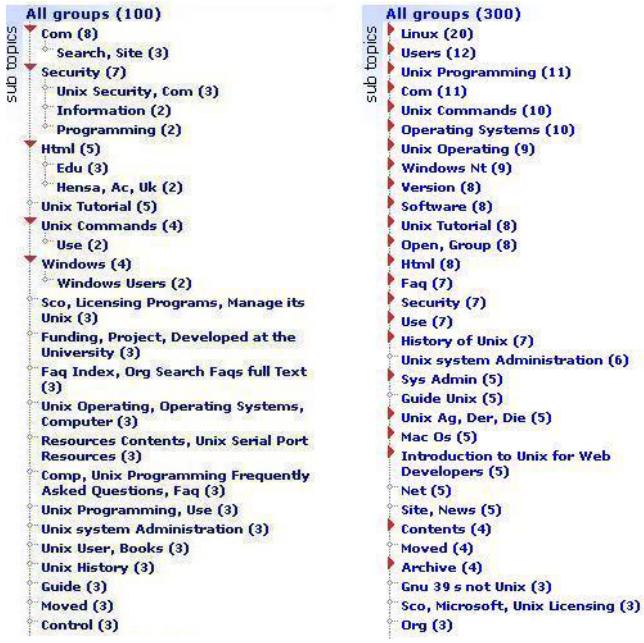

Fig. 6-3-2-1 Comparison of AHC results for a query "unix" using 100 (on the left) and 300 (on the right) input snippets.

The most important difference caused by increase of input snippets number are increase of number of created top-level clusters and growth of hierarchy, which has become much more complex (unfortunately, due to its size we were not able to present here the whole tree). It is clearly visible that while when using 300 snippets a dozen or so top-level groups have their own subclusters, in case of 100 snippets the structure is more flat, with hierarchy present only in few biggest clusters. However, increase of the groups number was much smaller than of the documents number due to repeating topics in the input set.

It turns out that our assumption about influence of increase in input data size on cluster labels quality is only partially fulfilled. Although some of the groups in the right tree have shorter descriptions, it is hard to say, whether their quality has really improved. It results most probably from the fact that problem with overspecialized labels in case of clusters created by the AHC algorithm is rather rare (compare LINGO). On the other hand, it is clearly visible that in hierarchy constructed by clustering a larger number of documents first clusters (i.e. the biggest ones) correspond better to the most important subtopics (due to the fact that algorithm is performed on a more representative sample).

6.3.3. INFLUENCE OF PHRASES USE

Purpose of this experiment was to show the profits of phrases use in the AHC algorithm and the vector space model in general. According to our hypothesis formulated in Chapter 3-5, use of phrases shouldn't have strong influence on structure of the hierarchy itself, but rather on quality of cluster labels.

Fig. 6-3-3-1 Comparison of AHC results for a query "king" for maximum length of extracted phrases equal to 1 (i.e. no phrases use – on the left) and 5 (on the right).

After closer examination of the tree structure it is easy to see that in spite of phrases use its structure has changed only slightly. This is due to the fact that if a group of documents shares a phrase, it also shares all of words that are included in it, so introduction of phrases should have no great impact on inter-document similarities. On the other hand, the clusters labels have become much more coherent. When we use only single terms, often their sequence in descriptions is different than that in documents, and this fact may confuse the user. Moreover, terms equal to the user's query are removed during terms weigthing from the term-document matrix (see Chapter 3-2-2) and therefore they cannot appear in labels, which become often too short and not fully informative. However, when using phrases the same terms may appear as their elements and therefore also in descriptions. This problem is illustrated on the above figure where labels of clusters such as *"King County"* or *"Burger King"* become *"County"* or *"Burger"* when we use single terms only.

6.3.4. INFLUENCE OF GROUPS CREATING THRESHOLD

Groups creating threshold parameter indicates, whether any groups should be created from the nodes of dendrogram, whose similarity is lower than value of this parameter (see Chapter 4-3-1). Default value of this parameter is equal to 0.0 so no groups are discarded only on the basis of their similarity level. We have supposed that setting this threshold to some higher number (in our experiment we have chosen 0.3) should leave in the results only the most coherent groups, moreover cleaned from snippets that don't fit well into them. The trade-off would of course be moving these documents to the *"Other Topics"* group, but we were willing to pay it as the main problem with the AHC algorithm consists rather in constructing weakly associated and poorly labeled groups than in its inability to assign documents to some cluster (compare with results of LINGO evaluation – [Osiński 03]).

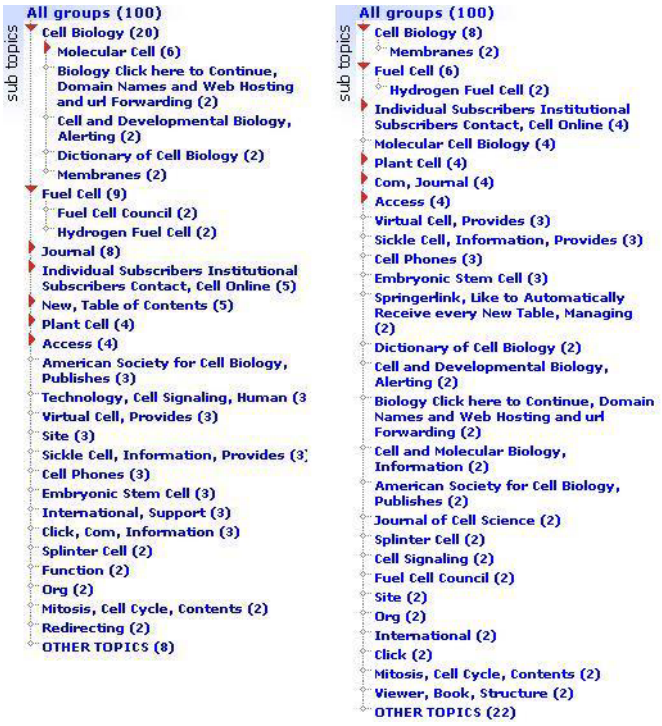

Fig. 6-3-4-1 Comparison of AHC results for a query "cell" using groups creating threshold parameter values of 0.0 (on the left) and 0.3(on the right).

It turned out that change of this parameter's value from 0.0 to 0.3 was enough to cause a sharp decrease in size of created clusters, without substantial changes in their quality. It is easy to see that in spite of using these different parameter values, descriptions of created clusters are almost the same, only their sizes have changed. One may also notice that the *"Other Topics"* group is much bigger and contains 22 instead of 8 documents. We may therefore say that increasing value of this threshold is not a good way of improving clustering quality.

6.3.5. INFLUENCE OF POST-PROCESSING METHODS

In Chapter 4-3-5 we have proposed dendrogram pruning methods based on descriptions of created clusters, and not on values of similarities between its nodes. Here we will show how their use improves the quality of results.

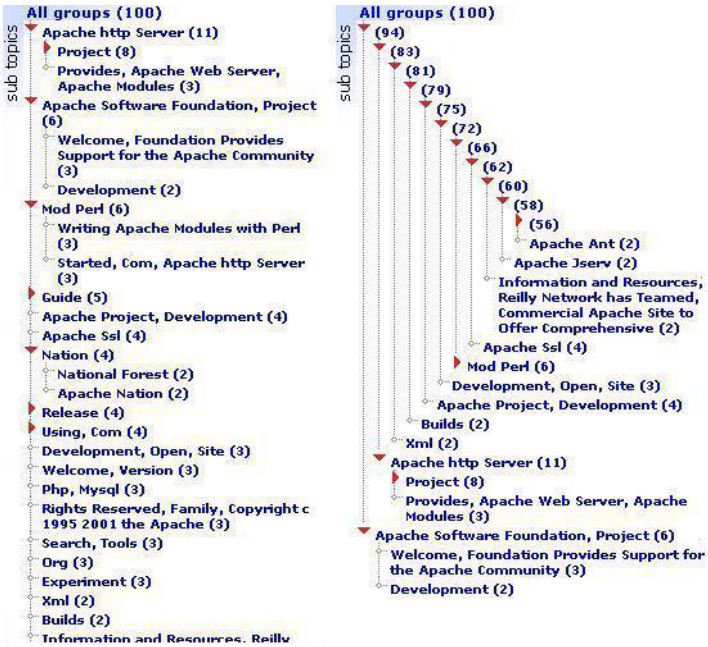

Fig. 6-3-5-1 Comparison of AHC results for a query "apache" with (on the left) and without (on the right) use of dendrogram post-processing strategies described in Chapter 4-3-5.

It is clearly visible that according to our hypothesis removing groups that have no descriptions has an essential influence on the quality of algorithm's results. In above example hierarchy consists of a dozen or so top level groups. As AHC creates a binary dendrogram, in order to present such hierarchy it has to introduce "artificial" groups into the results structure (compare to Figure 4-3-1-2). Character of these groups is their lack of descriptions, clearly visible on right tree in the above figure. Left dendrogram is created just by removing these clusters. Remaining post-processing methods from Chapter 4-3-5 are used quite rare, as situations when they may be used are also rare.

Another interesting observation is lack of the *"Other Topics"* group in the right tree. It is due to the fact that no clusters were removed from it and so there was no reason to create such special group for documents that would be contained in these groups.

6.3.6. INFLUENCE OF TERMS WEIGHTING METHODS

In this experiment we have confronted four already mentioned in Chapter 3-2-1 methods of terms weighting:

- standard tf terms weighting
- its version assigning the square root of the *tf* coefficient value as weight of a term
- standard tf-idf terms weighting (see Figure 3-2-1-3)
- its version proposed by us, using linearly decreasing *idf* factor (see Figure 3-2-1-4)

Dendrograms obtained by using these methods during terms weighting phase are shown below:

Fig. 6-3-6-1 Comparison of AHC results for a query "clinton" with different terms weighting methods used (from the left ot the right: tf, $\sqrt{tf}$, tf-idf with idf factor from Figure 3-2-1-3 and tf-idf with idf factor from Figure 3-2-1-4.

One may easily notice that it is hard to find any significant differences between hierarchies created by using these methods (with the sole exception of "standard" tf-idf weighting). In particular, ones created by using tf weighting and tf-idf formula from Figure 3-2-1-4 are very similar.

The most important difference occurring in case of use of standard tf-idf terms weighting method is frequently its inability to form larger clusters. In this example, this property manifests itself in case of the group *"Clinton County"* (being the largest from all created for used query). When using this method, this cluster is only half of its size when compared to use of other ones. This is due to our hypothesis from Chapter 3-2-1, where we have noticed that *idf* factor used in this method decreases too quickly with increase in number of documents in which a term appears.

It is hard to determine unequivocally, which one of the remaining methods allows to produce best results. In our opinion making such judgments would require conducting an experiment similar to one form Chapter 6-2, with at least several different queries and exact quality measures used. It is needless to say that in fact there would have to be *three* experiments – one for each of the presented methods.

6.4. EXPERT EVALUATION OF THE AHC ALGORITHM - COMPARISON OF AHC AND OTHER CLUSTERING ALGORITHMS

In this subchapter we present a rough comparison of Agglomerative Hierarchical Clustering and three other algorithms:

- hierarchical Suffix Tree Clustering [Masłowska and Słowiński 03], creating the clusters tree on the basis of results of applying of "classical" STC algorithm (refer to Chapter 4-1-3 for details)
- hierarchical method used in the Vivisimo clustering service [Vivisimo]. Unfortunately, authors of this system don't reveal any secrets about type of used algorithm
- the LINGO algorithm [Osiński 03], being also an element of the Carrot2 system. It is the only compared here method that doesn't create a hierarchy.

It would be also interesting to compare "our" AHC implementation and some other version of this algorithm. However, unfortunately we don't know about any available implementation, so we couldn't perform such comparison.

The experiment was conducted on the results of a *"data mining"* query from the Vivisimo service. 400 top snippets (after duplicate removal about 350) were downloaded and then clustered using different algorithms. Such large input set was used in order to fairly compare these methods, as hierarchical STC was designed especially for clustering of big numbers (i.e. hundreds) of documents.

Fig. 6-4-1 Comparison of results of different clustering algorithms for a query "data mining". From left to right: AHC, LINGO, Vivisimo, and hierarchical STC (results of all methods are visualized using Carrot^2 output component). Different total numbers of snippets in case of different algorithms result from overlapping.

After a closer examination one may notice that results created by different algorithms share a lot of common clusters (this is true especially in case of the largest ones). Groups such as labeled with phrases *"Knowledge Discovery"*, *"Data Warehouses"*, *"Machine Learning"*, *"Visualization"* appear in all hierarchies (although their exact descriptions sometimes differ between results of different algorithms).

A noteworthy fact in case of AHC results is very small size of the "Other Topics" group (consisting only of 1 document of 350 present in the input). We have also noticed that results given by this algorithm in case of snippets downloaded from Vivisimo are slightly better than when using Google as data source. This may be due to the fact that Vivisimo makes use of several search engines that also probably provide longer snippets.

Here we will point out the most important properties of results of AHC and other algorithms:

- AHC

  Agglomerative Hierarchical Clustering creates very small clusters when compared to the other methods. This is partially due to the fact that it is unable to produce overlapping results. Moreover, the clusters are labeled with very general descriptions (in part because of their hierarchical structure – groups with long, specific descriptions occur in the lower levels of hierarchy). These descriptions may be too short to understand the topic of the group (for instance *"Text"* or *"Offers"*).

- LINGO

  This method creates the smallest number of clusters from all algorithms presented here. On the other hand, their descriptions are very long and meaningful (although sometimes unfortunately *too long* and not matching all documents in their respective groups). For instance, groups labeled by the other methods *"Decision Support"* or *"Notes"* are described by LINGO respectively with phrases *"Decision Support Software"* and *"Data Mining Lecture Notes"*.

- Vivisimo

  Results of method employed in Vivisimo seem to have no "outstanding features". In all considered aspects (size and number of groups and length of their descriptions), this algorithm seems to find a "compromise" between AHC and LINGO. One may also notice that hierarchy produced by it is less deep than one created by the AHC algorithm.

- hierarchical STC

  The most characteristic property of results of this method is their strong overlapping (compare total sizes of created clusters on Figure 6-4-1). Hierarchical STC creates several huge groups, but also a lot of very small, so that its results consist of biggest number of clusters when compared to ones of other algorithms (unfortunately we were unable to present on figure the whole list of top-level groups due to its length). Moreover, labels of top-level clusters are very short (shorter even than in case of AHC), probably due to the same reasons as in case of this method.

7. CONCLUSIONS

Web mining, and in particular search results clustering are relatively new and quickly developing domains of computing science. These domains are promising and certainly there still remains a lot of work to be done before the clustering algorithms will be widely used. New approaches to this problem are continually developed (for instance LINGO, which was created during the work on the Carrot^2 system). Current methods usually seem to have a great advantage over old algorithms derived from the areas of both statistics and text databases (such as Agglomerative Hierarchical Clustering, which is the main subject of our thesis), as they pay more attention to meaning of texts contained in the input data, and not only to simple common occurrences of particular words in them.

However, as we have shown during the work on this thesis, thanks to introduction of some changes adapting it to text processing, also the AHC algorithm may successfully perform documents clustering. Results collected during the algorithm's evaluation clearly show that it may be useful in our application. It shouldn't be forgotten that AHC is algorithm designed rather for offline clustering, where this activity may be performed on larger collections of complete documents. Here we have shown that its results may be helpful to users even if the input data set is small and consists only of *digests* of documents.

This is an effect of employing pre- and post-processing methods that enable to fit this algorithm to our application domain. Thanks to use of these methods (i.e. phrases extraction and dendrogram pruning) it has become more sensitive to *meaning* of clustered texts. Our research has been guided towards these techniques also due to the fact that AHC itself is a well-studied and used for tens of years algorithm. But pre- and post-processing phases that according to us are very important are considerably less studied. Moreover, we have found these phases, where not numerical, but textual data is processed much more interesting.

Finally, we would like to emphasize meaning of the fact that this thesis is a part of much greater whole. Our implementation of AHC is a module of the Carrot^2 system, which is the most advanced, if not the only one Open Source clustering service. Due to the fact of its integration in the Carrot^2 framework, software being the result of this thesis may turn out much more useful than if it were a separate application. Moreover, maybe in the future our module will be developed further in order to perform research on other aspects of the algorithm.

7.1. SCIENTIFIC CONTRIBUTIONS

Main scientific contributions of this thesis include:

- use of phrases in the AHC algorithm

  We have succeeded in implementing of a simple statistical technique of phrases extraction and employing it in our module. However simple, this method turned out to

give useful results. Moreover, it can be easily applied in any algorithm using the vector space model.

- introduction of description-based dendrogram pruning methods

  In order to perform post-processing of the algorithm's results we have proposed techniques based on analysis of result clusters labels instead of ones based on analysis of similarities between them, as we feel that these methods are more "topic-aware". This approach has been introduced also according to repeatedly emphasized in publications from this domain importance of good cluster descriptions.

- implementation of a hierarchical clustering algorithm as a part of the Carrot^2 system

  During work on this thesis we have implemented the AHC algorithm together with mentioned before methods. Then our code has been released as a part of Open Source Carrot^2 system. This module is the only one from three clustering components in this system capable of generating hierarchical results.

- simple evaluation of our version of the algorithm performed

  We have conducted different types of evaluation experiments – user- and expert-oriented. User assessment has shown that our implementation of the AHC algorithm gives useful results. On the other hand, expert evaluation enabled us to study the influence of different parameters of the algorithm on its results and compare AHC and recent approaches to document clustering.

7.2. POSSIBLE FUTURE RESEARCH

Unfortunately, we haven't succeeded in addressing all issues associated with implementation of AHC. However, we are aware of existence of these problems and point them out here as subjects of possible future research:

- extension of input pre-processing

  Methods of pre-processing of the input snippets, performed by the other modules of the Carrot^2 system, such as stemming or stop-words removal also have a strong influence on the final results. We feel that for instance extension of stop-lists would result in a considerable improvement of result clusters quality as descriptions of clusters created by our algorithm often consist of single terms only. This fact makes them unclear in case when these terms are ones which contain no information and should in fact be put on the stop-list.

- overlapping clusters

  Unfortunately, in results given by AHC one document may appear only in one group (not counting its parent groups in hierarchy). Yet in the real world single documents may correspond to several different topics and should be contained in an appropriate number

of clusters. However, during our work on this thesis we haven't found any publications mentioning attempts to create version of the AHC algorithm with such capabilities.

- improvement of usefulness of created clusters

We feel that main problem with the quality of AHC results is that it quite often creates clusters made up from documents which share just one or two single words, which usually don't tell us anything about their real *topic*. So rejecting some of the created clusters and moving documents contained in them to the *"Other Topics"* group may have in fact an overall good influence on the results.

- speed of implementation

Unfortunately, our implementation of the AHC algorithm is certainly too slow for its practical application. Main reason of this fact is the cubic complexity of used version of the clustering algorithm itself.

8. BIBLIOGRAPHY

[AltaVista] *AltaVista indexing service*, http://www.altavista.com

[AnswerBus] *AnswerBus search engine with a natural language interface*, http://misshoover.si.umich.edu/~zzheng/qa-new

[AskJeeves] *AskJeeves search engine with a natural language interface*, http://www.askjeeves.com

[Bezdek 81] Bezdek, J. C.: *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, New York, 1981

[Carrot] *Carrot Web search results clustering interface*, http://www.cs.put.poznan.pl/dweiss/index.php/research/carrot/obsolete.xml

[Carrot2] *Carrot^2 Web search results clustering interface*, http://www.cs.put.poznan.pl/dweiss/carrot

[Chabiński and Bugajska 03] Chabiński A., Bugajska M.: *Multiszperacze*, CHIP, 118 (3): 128-132, March 2003

[Church et al. 91] Church, K.W., Gale, W., Hanks, P., Hindle, D.: *Using Statistics in Lexical Analysis,* in: Zernik, U. (ed.): *Lexical Acquisition: Exploiting On-Line Resources to Build a Lexicon*: 115-164, Lawrence Erlbaum, New Jersey, 1991

[Cutting et al. 92] Cutting, D. R., Karger, D. R., Pedersen, J. O., Tukey, J. W.: *Scatter/Gather: a cluster-based approach to browsing large document collections*, Proceedings of the 15th International ACM SIGIR Conference on Research and Development of Information Retrieval (SIGIR'92): 318-329, 1992.

[Dogpile] *Dogpile meta-search engine*, http://www.dogpile.com

[Dom 2001] Dom, E. B., *An information-theoretic external cluster validity measure*, IBM research report RJ 10219, 2001.

[Duff et. al 02] Duff, I. S., Heroux, M. A., Pozo R., *An overview of the sparse basic linear algebra subprograms: The new standard from the BLAS technical forum*, ACM Transactions on Mathematical Software (TOMS), 28 (2): 239-267, June 2002.

[Egothor] *Egothor indexing service*, http://somis4.ais.dundee.ac.uk:8080/egothor-api

[Emulti] *Emulti meta-search engine*, http://www.emulti.pl

[Everitt 80] Everitt, B.: *Cluster Analysis*. Halsted Press (John Wiley & Sons), New York, 1980

[Google] *Google indexing service*, http://www.google.com

[Hill 68] Hill, D. R.: *A vector clustering technique*, in: Samuelson (ed.): *Mechanized Information Storage, Retrieval and Dissemination*, North-Holland, Amsterdam, 1968.

[Java] *The source for Java Technology*, http://java.sun.com

[Kartoo] *Kartoo graphical search results visualization service*, http://www.kartoo.com

[Karypis and Han 00] Karypis, G., Han, E-H., *Concept Indexing A Fast Dimensionality Reduction Algorithm with Applications to Document Retrieval & Categorization*, Technical Report TR-00-0016, University of Minnesota, 2000

[Lance and Williams 66] Lance, G. N., Williams, W. T.: *A General Theory of Classificatory Sorting Strategies. 1. Hierarchical Systems*, Computer Journal, 9: 373-380, May 1966.

[LookSmart] *LookSmart Web directory*, http://www.looksmart.com

[Lovins 68] Lovins, J. B.: *Development of a Stemming Algorithm*, Mechanical Translation and Computational Linguistics, 11(1): 23-31, March 1968

[Maarek et al. 91] Maarek, Y., Berry, D. M., Kaiser G. E.: *An Information Retrieval Approach For Automatically Constructing Software Libraries*, IEEE Transactions On Software Engineering, 17 (8): 800-813, August 1991

[Maarek et al. 00] Maarek, Y., Fagin R., Ben-Shaul, I., Pelleg D.: *Ephemereal Document Clustering for Web Applications*, IBM Research Report RJ 10186, April 2000.

[MacQueen 67] MacQueen, J.: *Some methods for classification and analysis of multivariate observations*. Proceedings of the Fifth Berkeley Symposium on Mathematical statistics and probability, vol. 1: 281-297, University of California Press, Berkeley, 1967.

[Mamma] *Mamma meta-search engine*, http://www.mamma.com

[MapNet] *MapNet graphical search results visualization service*, http://maps.map.net

[Masłowska and Słowiński 03] Masłowska, I., Słowiński, R., *Hierarchical Clustering of Text Corpora Using Suffix Trees*, in: Kłopotek, M.A., Wierzchoń, S.T., Trojanowski, K. (eds.): *Intelligent Information Processing and Web Mining*, *Advances in Soft Computing*, 179-188, Springer-Verlag, 2003

[Metasearch] M*etasearch meta-search engine*, http://www.metasearch.com

[MSN] *MSN indexing service*, http://www.msn.com

[NorthernLight] *NorthernLight Web directory*, http://www.northernlight.com

[Notess 99] Notess, G.: *Dead Links Report*,
http://www.searchengineshowdown.com/stats/dead.shtml

[ODP] *Open Directory Project Web directory*, http://dmoz.org

[Osinski 03] Osiński, S.: *An Algorithm for Clustering of Web Search Results*, Master thesis, Poznań University of Technology, 2003

[Page et al. 98] Page, L., Brin, S., Motwani, R., Winograd, T.: *The Page Rank citation ranking: Bringing order to the Web*, Technical Report, Stanford University, 1998

[Porter 80] Porter, M. F.: *An algorithm for suffix stripping*, Program, 14(3): 130-137, 1980

[RazDwaTrzy] *RazDwaTrzy meta-search engine*, http://razdwatrzy.com

[Rocchio 66] Rocchio, J. J.: *Document retrieval systems – optimization and evaluation*, Ph.D. thesis, Harvard University, 1966.

[Salton 89] Salton, G.: *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*, Addison-Wesley, 1989

[Salton and Buckley 87] Salton, G., Buckley, C.: *Text Weighting Approaches in Automatic Text Retrieval*, Cornell University Technical Report: 87-881, New York, 1987

[Salton et al. 75] Salton, G., Wong, A., Yang, C. S.: *A Vector Space Model for Automatic Indexing*, Communications of the ACM, 18 (11): 613-620, November 1975

[SearchEngineWatch] *Web service containing lots of information about search engines*, http://www.searchenginewatch.com

[Selberg 99] Selberg, E. W.: *Towards Comprehensive Web Search*, Doctoral dissertation, University of Washington, 1999

[Smadja 91] Smadja, F. A.: *From N-Grams to Collocations: An Evaluation of Xtract*, Proceedings of 29th ACL Meeting, Berkeley, 1991

[Smadja 93] Smadja, F. A.: *Retrieving collocations from text: Xtract*, Computational Linguistics, 19(1): 143—177, 1993

[Stefanowski and Weiss 03] Stefanowski, J., Weiss, D., *Carrot^2 and Language Properties in Web Search Results Clustering*, Proceedings of the First International Atlantic Web Intelligence Conference (AWIC'2003), 240-249, Madrid, Spain, 2003

[Ukkonen 95] Ukkonen, E.: *On-line construction of suffix trees*, Algorithmica, 14(3): 249-260, September 1995

[UML] *UML Resource Page*, http://www.omg.org/technology/uml

[vanRijsbergen 79] van Rijsbergen, C. J.: *Information Retrieval*, Butterworths, London, 1979

[Vivisimo] *Vivisimo Web search results clustering interface*, http://www.vivisimo.com

[Voorhees 86] Voorhees, E. M: *Implementing agglomerative hierarchical clustering algorithms for use in information retrieval*, Information Processing and Management, 22: 465-476, 1986

[Weiss 01] Weiss, D.: *A Clustering Interface for Web Search Results in Polish and English*. Master thesis, Poznań University of Technology, 2001

[Weiss 02a] Weiss, D., Carrot Developers: *Carrot^2 Developers Guide*, http://www.cs.put.poznan.pl/dweiss/carrot/index.php/developers, 2002

[Weiss 02b] Weiss, D.: *Szukanie igły w sieci*, Magazyn Internet, 80 (5): 46-51, May 2002

[Weiss 02c] Weiss D.: *Choć na chwilę zdjąć gogle..*, CHIP, 112 (9): 130-134, September 2002

[Weiss and Stefanowski 03] Weiss, D., Stefanowski, J.: *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, vol. 579 (XIV), 209-220, Zakopane, Poland, 2003.

[WordNet] *WordNet linguistic system*, http://www.cogsci.princeton.edu/~wn

[Yahoo] *Yahoo indexing service*, http://www.yahoo.com

[Zamir 99] Zamir, O., *Clustering Web Algorithms: A Phrase-Based Method For Grouping Search Engine Results*, Doctoral Dissertation, University of Washington, 1999.

[Zamir and Etzioni 98] Zamir, O., Etzioni, O.: *Web Document Clustering: A Feasibility Demonstration*, Proceedings of the 19th International ACM SIGIR Conference on Research and Development of Information Retrieval (SIGIR'98): 46-54, 1998.

[Zamir and Etzioni 99] Zamir, O., Etzioni, O.: *Grouper: A Dynamic Clustering Interface to Web Search Results*. WWW8 / Computer Networks 31(11-16): 1361-1374, 1999